Pedro Miguel David Rechena

Licenciado em Ciências da Engenharia Electrotécnica e de Computadores

Comparison of anomaly detection techniques applied to different problems in the telecom industry

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientador: Luís Bernardo, Associate Professor, NOVA University of Lisbon

Co-orientadora: Sabina Zejnilović, Ph.D.

Júri

Presidente: Doutor Nuno Filipe Silva Veríssimo Paulino - FCT/UNL

Arguentes: Doutor João Carlos Gomes Moura Pires - FCT/UNL

Vogais: Doutor Luís Filipe Lourenço Bernardo - FCT/UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

November, 2020

Comparison of anomaly detection techniques applied to different problems in the telecom industry

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

Many people influenced this journey and help me along the way, and this dissertation could not have been written without all of them.

I would like to thank my advisor, professor Luís Bernardo, for the support given throughout my course and especially during the writing of this dissertation, for all the availability shown and and for the advice given.

I want to leave a special thanks to my co-advisor Sabina Zejnilović, for giving me the opportunity of developing my dissertation as part of this project and letting me be a part of a great team during these last months. But mainly for all the help given in the duration of the project, the guidance throughout its implementation, the sharing of knowledge, the availability demonstrated and for all the patience. I would also like to thank the members of the team that I was part of, for the help and for the teachings that enabled me to complete this project.

Then, I would like to thank my friends for being there for me in the good and bad moments of my life, and for all the support that they gave me during these last months and also thank all my colleagues for these great years of college.

Last but not least, I want to thank my family that gave me the opportunity to study and supported me during all these years, and also for giving me the motivation I needed to succeed.

Whether you believe you can do a thing or not, you are right.
Henry Ford

Abstract

Nowadays, with the growth of digital transformation in companies, a huge amount of data is generated every second as a result of various processes. Often this data contains important information which, when properly analyzed, can help a company gain a competitive advantage. One data processing task common to many different applications is detection of anomalies, that is, data points or groups of data points that stand out from most of the others. Since it is not feasible to have an operator constantly analyzing the data to find anomalous values, due to the generally large volumes of data, the focus of this dissertation is the exploration of a Data Mining area called anomaly detection. In this dissertation we first develop an anomaly detection software in Python, that applies 10 different anomaly detection algorithms, after automatically optimizing their parameters, to an arbitrary dataset. Before applying these algorithms, the software also performs the task of data scaling and imputation of missing values. It outputs the results of the performance metrics of each algorithm, the values of the optimized parameters and the graphics for the results visualization generated using the method t-SNE. This software was then applied to three case studies to compare the performance of different anomaly detection approaches using real-world datasets. These datasets have an increasing level of difficulty associated with them: the amount of missing data and the uncertainty associated with the ground truth regarding the anomalies. In the first case study, we detected fraudulent bank transactions using a public dataset. Then, in the second case we identified clients of a telecommunication company who were likely to miss their payment, leading to contract termination. For this case we used a dataset from a telecommunications company. In the third case, we detected low quality of internet service, again using a large dataset with real measurements from a telecommunications company. Finally, we implemented a state of the art, neural network model, specially applicable to the task of identifying anomalies in time-series data. We optimized the parameters of the network, and applied it to address the problem of low quality of service.

Keywords: Anomaly Detection, Machine learning, Unsupervised Learning, Time series, LSTM

Resumo

Com o crescimento da transformação digital nas empresas, uma quantidade enorme de dados são gerados a cada segundo como consequência de variados processos. Muitas das vezes esses dados contêm informação importante que podem permitir a uma determinada empresa obter uma vantagem competitiva. Uma forma de obter conhecimento sobre o actual funcionamento de um determinado processo é através da detecção de anomalias, ou seja, instâncias de dados que se destacam da maioria das restantes. Visto não ser viável ter um operador a visualizar linhas de dados para encontrar anomalias, devido às dimensões dos dados, o foco desta dissertação revolve em torno da exploração de uma área de *Data Mining* chamada detecção de anomalias. Nesta dissertação propõe-se em primeiro lugar um *software* de detecção de anomalias feito em *Python* que aplica um conjunto de 10 algoritmos de detecção de anomalias, depois de optimizar os seus parâmetros automaticamente, a um conjunto de dados arbitrários. Antes da aplicação dos algoritmos, o *software* realiza primeiramente a sua normalização e a imputação dos valores nulos. Por fim, retorna os resultados das métricas de desempenho de cada algoritmo, os parâmetros escolhidos e um conjunto de gráficos para visualização de resultados, gerados utilizando t-SNE. Este software foi então aplicado a três casos de estudo para comparar o desempenho das diferentes técnicas utilizando conjuntos de dados reais. Estes conjuntos de dados têm um nível crescente de dificuldade associado a eles: a quantidade de valores nulos e a incerteza em relação aos pontos realmente anómalos. O primeiro é relacionado com transacções bancárias onde se utilizou um conjunto de dados público. Depois, um caso de estudo relacionado com cessações de contrato devido à falta de pagamento, onde foi utilizado um conjunto de dados de uma empresa de telecomunicações. Por último um caso de estudo relacionado com a qualidade de serviço de clientes de uma empresa de telecomunicações. Por fim, foi implementada uma arquitectura de um modelo de redes neuronais avançado de detecção de anomalias em séries temporais, que foi utilizado para detectar anomalias no conjunto de dados de qualidade de serviço.

Palavras-chave: Detecção de anomalias, *Machine learning*, Aprendizagem não supervisionada, Séries temporais, *LSTM*

Contents

List of Figures xvii

List of Tables xix

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 Objectives . . . 2
- 1.3 Contributions . . . 2

2 State of Art 3
- 2.1 Introduction . . . 3
- 2.2 The Anomaly Detection Problem . . . 4
  - 2.2.1 Input Data . . . 4
  - 2.2.2 Type of Anomalies . . . 6
  - 2.2.3 Output of Anomaly Detection . . . 6
- 2.3 Anomaly Detection Algorithms . . . 7
  - 2.3.1 Categorization of Algorithms . . . 7
  - 2.3.2 Algorithms for Tabular Data . . . 11
  - 2.3.3 Algorithms for Time Series Data . . . 15
- 2.4 Performance Metrics . . . 16
  - 2.4.1 Fundamentals . . . 16
  - 2.4.2 Evaluation Measures . . . 17
- 2.5 Data Mining Process . . . 19
  - 2.5.1 Business Understanding . . . 20
  - 2.5.2 Data Understanding . . . 20
  - 2.5.3 Data Pre-Processing . . . 21
  - 2.5.4 Modeling . . . 23

3 Description of Developed Software 25
- 3.1 Data Pre-Processing Steps . . . 26
  - 3.1.1 Dealing with Missing Data . . . 26
  - 3.1.2 Selecting Outlier Percentage . . . 27
  - 3.1.3 Scaling of Data . . . 27

3.1.4 Choosing t-SNE Perplexity . . . 28
3.2 Algorithm Implementation Details . . . 28
3.2.1 Isolation Forest . . . 28
3.2.2 Extended Isolation Forest . . . 29
3.2.3 Local Outlier Factor . . . 29
3.2.4 One Class Support Vector Machine . . . 30
3.2.5 Elliptic Envelope . . . 31
3.2.6 Fast Angle-Based Outlier Detection . . . 31
3.2.7 K-Nearest Neighbor . . . 31
3.2.8 Gaussian Mixture Models . . . 32
3.2.9 Auto-Encoder . . . 32
3.2.10 Principal Component Analysis . . . 33

4 Comparison of Anomaly Detection Algorithms **35**
4.1 Detecting Credit Card Fraud . . . 35
4.1.1 Data Pre-Processing . . . 36
4.1.2 Data Visualization . . . 37
4.1.3 Results . . . 37
4.2 Detecting Involuntary Churners . . . 46
4.2.1 Data Pre-Processing . . . 47
4.2.2 Data Visualization . . . 47
4.2.3 Results . . . 48
4.3 Conclusions . . . 58

5 Detecting Low Quality of Service **59**
5.1 Quality of Service Dataset . . . 60
5.2 Classical Anomaly Detection in the Context of QoS . . . 61
5.2.1 Data Pre-Processing . . . 61
5.2.2 Data Visualization . . . 62
5.2.3 Algorithm Performance . . . 63
5.3 LSTM Based Anomaly Detection in the Context of QoS . . . 67
5.3.1 LSTM Basic Architecture . . . 67
5.3.2 Results . . . 74
5.3.3 Impact of Contextual Variables and Treatment of Missing Values . . . 76
5.4 Conclusions . . . 81

6 Conclusion and Future Work **83**
6.1 Conclusion . . . 83
6.2 Future Work . . . 85

Bibliography **87**

A t-SNE plots with predictions of the anomaly detection software algorithms 91

List of Figures

2.1 Example of Anomalies in 2D Data [3] . . . 3
2.2 Auto-Encoder Architecture [15] . . . 14
2.3 Example of a Confusion Matrix . . . 17
2.4 CRISP-DM Process Model for Data Mining [37] . . . 20

3.1 Anomaly Detection Software Architecture . . . 26

4.1 Correlation Matrix from credit card fraud training dataset . . . 36
4.2 Credit card fraud t-SNE with real labels . . . 37
4.3 Credit card fraud tuning gain . . . 39
4.4 Comparison of Gaussian Mixture Models predictions between the default and tuned versions in credit card fraud dataset . . . 40
4.5 Comparison between the confusion matrix of Gaussian Mixture Models and Fast ABOD in the credit card dataset . . . 43
4.6 Comparison between the t-SNE plots of Gaussian Mixture Models and Fast ABOD for the credit card dataset . . . 43
4.7 Credit card fraud ROC Curves . . . 44
4.8 Credit card fraud PR Curves . . . 44
4.9 Comparison of lift plots between the Isolation Forest and Gaussian Mixture Models predictions in the credit card dataset . . . 45
4.10 Comparison of t-SNE plots between the Isolation Forest and Gaussian Mixture Models predictions in the first percentile on the credit card dataset . . . 46
4.11 Involuntary churn t-SNE with the real labels and the different clusters observed 49
4.12 Involuntary churn tuning gain . . . 49
4.13 Comparison of Local Outlier Factor predictions between the default and tuned versions in the involuntary churn dataset . . . 51
4.14 Comparison of Elliptic Envelope predictions between the default and tuned versions . . . 52
4.15 Comparison of Gaussian Mixture Models predictions between the default and tuned versions in involuntary churn dataset . . . 52
4.16 Comparison between the confusion matrix of Local Outlier Factor and Auto-Encoder in the involuntary churn dataset . . . 55

4.17 Comparison between the t-SNE plots of Local Outlier Factor and Auto-Encoder for the involuntary churn dataset . . . 56
4.18 Involuntary churn ROC Curves . . . 56
4.19 Comparison of lift plots between the Local Outlier Factor and Extended Isolation Forest predictions . . . 57
4.20 Comparison of t-SNE plots between the Local Outlier Factor and Extended Isolation Forest predictions in the first percentile . . . 58

5.1 t-SNE plots of the QoS dataset . . . 63
5.2 Comparison of the confusion matrices of One-Class SVM and PCA for the QOS dataset . . . 66
5.3 t-SNE plots of the results of One-Class SVM and PCA for the QoS dataset . . . 67
5.4 Input shape of an LSTM neural network [22] . . . 68
5.5 Transforming the dataset into LSTM input shape [35] . . . 68
5.6 LSTM Auto-Encoder metrics with timestep variation . . . 69
5.7 Comparison of the LSTM Auto-Encoder losses using 7 and 21 timesteps . . . 70
5.8 LSTM Auto-Encoder Anomaly Score with different Time Steps . . . 71
5.9 LSTM Auto-Encoder metrics with batch variation . . . 71
5.10 LSTM Auto-Encoder metrics with architecture variation . . . 72
5.11 LSTM encoder architecture [36] . . . 73
5.12 LSTM decoder architecture [36] . . . 74
5.13 LSTM Auto-Encoder results for the quality of service dataset . . . 75
5.14 Example of customer with an anomaly found in the hours of complaint . . . 76
5.15 Resultant anomaly score and threshold when using the contextual information of the day of the week . . . 79
5.16 Resultant anomaly score and threshold by using the contextual information of the day of the week and hour . . . 79
5.17 Resultant anomaly score and threshold by using the contextual information of the customer's CMTS . . . 80
5.18 Resultant anomaly score and threshold by using imputation of missing values by mean and addition of flag . . . 81

A.1 t-SNE plots with the predictions of the tuned algorithms for the credit card dataset . . . 91
A.1 t-SNE plots with the predictions of the tuned algorithms for the credit card dataset . . . 92
A.2 t-SNE plots with the predictions of the tuned algorithms for the involuntary churn dataset . . . 93

List of Tables

2.1 Algorithm Overview 15

4.1 Credit Card Dataset Overview 37
4.2 Fast ABOD default and tuned parameters in credit card fraud dataset 39
4.3 K-Nearest Neighbor Detector default and tuned parameters in credit card fraud dataset 39
4.4 Gaussian Mixture Models default and tuned parameters in credit card fraud dataset 40
4.5 Performance of different anomaly detection algorithms for the credit card fraud dataset 42
4.6 Involuntary Churn Dataset Overview 47
4.7 LOF default and tuned parameters in involuntary churn dataset 50
4.8 Elliptic Envelope default and tuned parameters in involuntary churn dataset 51
4.9 Gaussian Mixture Models default and tuned parameters in involuntary churn dataset 52
4.10 Performance of different anomaly detection algorithms for the involuntary churn dataset 54

5.1 Quality of Service Dataset Overview 62
5.2 Performance of different anomaly detection algorithms for the quality of service dataset 65
5.3 Comparison between LSTM-AE and three of the best algorithms from the anomaly detection software 76
5.4 Comparison of different LSTM-AE contextual configurations 78
5.5 Comparison of LSTM-AE results obtained by using different missing value imputation methods 80

Chapter 1

Introduction

1.1 Context

We live in a world flooded with data and with that comes a constant need to process and understand data quickly. One data processing task common to many different applications is anomaly detection. This is the process of finding points of data or collections of points in a given dataset, with a behavior very different from the expected [12]. These anomalous data points are called outliers or anomalies and sometimes they are caused by errors during the process of data collection. However, they can also be an important source of information that can inform us of a system mechanical failure, fraudulent behaviors, or human errors, that due to the large volumes of the data are impossible to detect by the human eye. Anomaly detection techniques have been applied in many different applications: fraud detection, intrusion detection, activity monitoring, network performance, fault diagnosis, time-series monitoring, etc [16].

One of the sectors that benefits from the application of anomaly detection techniques is the industry of telecommunications, where massive amounts of data are generated every day, containing vital information about the user experience of the customers. This information is processed to improve the services that the operator offers and to give the operator a competitive edge. A client that is likely to churn who stops using the operator services or a faulty router that starts malfunctioning could be identified by looking for atypical patterns in data.

Different methods can be used to detect these outliers, depending on the type of data and the type of outliers we wish to detect. Statistical, machine learning methods (Supervised, Unsupervised, Semi-Supervised), or a combination of both are some methods typically used for this task.

1.2 Objectives

The customers of a telecommunications company usually exhibit certain patterns in their typical behavior when interacting directly or indirectly with the company, so the process of detecting anomalous events on customer data, could help telecommunication companies in the early detection of problems troubling the customer's user experience and in that way prevent churn. The objective of this dissertation is the development of a state of the art anomaly detection approach to be applied to different customer-related problems, and also the development of a general software containing several anomaly detection algorithms, to aid the process of choosing the approach that better fits the problem in question.

1.3 Contributions

In this dissertation, we applied a wide range of anomaly detection methods to solve several different problems in a telecommunications company, such as churn prediction, identification of payment anomalies, and detection of intervals with low quality of service. This work does not only include the study and comparison of different algorithms used in anomaly detection, but it also includes the pre-processing work that has to be done before these algorithms can be applied to real-world datasets. These pre-processing steps include data exploration and understanding, as well as transforming the dataset into a form that can be used as an input to different anomaly detection algorithms.

The contributions of this thesis are threefold:

- We developed a software that facilitates the application of ten different anomaly detection approaches to an arbitrary dataset. The software also performs the pre-processing needed for optimizing algorithms performance: treatment of missing data, data scaling, as well as hyperparameter tuning of the algorithms.

- We performed an in-depth study of the performance of ten different algorithms used in the developed software, for three distinct problems in the telecom industry: fraud, churn, and quality of service. The datasets used for these problems have an increasing level of difficulty associated with them: the amount of missing data and the uncertainty associated with labels, i.e., ground truth regarding the outliers. We compare and discuss the algorithms' performance using different metrics, not only related to the accuracy but also regarding computational and training requirements.

- We applied a state of the art neural network model, analyzed and optimized its performance to identify anomalies in time series data, where some data is missing and there is uncertainty associated with labels.

Chapter 2

State of Art

2.1 Introduction

An anomaly or an outlier represents a specific data point or a collection of data points from a dataset, that is much different from the majority of the other points as if it has not been generated by the same process. An illustration is given in figure 2.1, where we can notice the difference between the groups of points N_1 and N_2, which correspond to the normal data points, and the group of points O_1 and the points O_2 and O_3, which correspond to the anomalies.

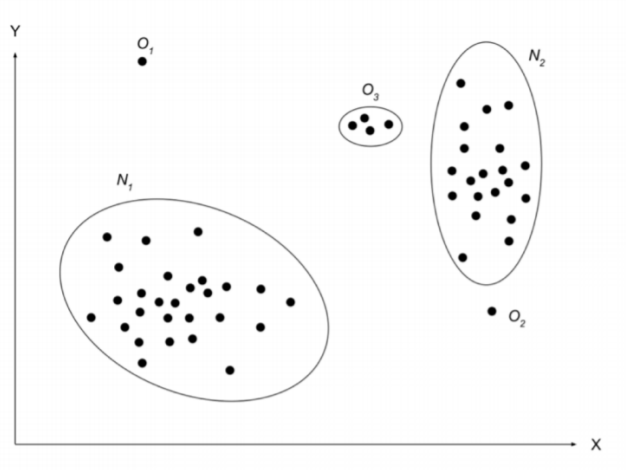


Figure 2.1: Example of Anomalies in 2D Data [3]

It is important to note that an anomaly is not the same as the noise in the dataset, because noise is usually caused by an error in the measuring device or a problem in the data storage unit and, unlike anomalies, does not contain additional relevant information.

Being able to find these anomalies can turn out to be a great advantage in various applications, but this task comes with a lot of challenges [4]:

- The task of modeling all the different patterns related to normal behavior can be difficult, mainly in dynamic systems.
- The normal behavior of the system can be time-varying, meaning that a normal pattern at this moment may not be normal in the future.
- Depending on the application domain of anomaly detection, the notion of anomaly may vary, so we cannot assume that an efficient technique in some fields is also efficient in others.
- The availability of labels in the data used by anomaly detection techniques is usually poor and when available tends to be inexact.
- It is very common for the data to be contaminated with noise and missing values, making it hard for the algorithms to distinguish them from the real anomalies.

2.2 The Anomaly Detection Problem

2.2.1 Input Data

One important step to understanding what type of anomaly detection algorithm to use is the nature of the input data such as the dimensionality of the dataset, the type of the attributes, and the relation between different data points [4].

2.2.1.1 Type of Attributes

Attributes or features are the columns of the datasets and represent a specific characteristic of a data point. All values of the same attribute have to be of the same type, but different attributes can have different types. These types can be nominal, binary, ordinal, or numeric [12, ch.2].

Nominal: The values of a nominal attribute are names and represent categories or states, so there is no order between them. The values can also be represented as numbers, but in this case, the numbers cannot be used quantitatively and do not have any order between them. One example of a nominal attribute is a type of service or a contract that a client might have with a company. In chapter 5 we use this type of variable as contextual information in a neural network by transforming the nominal values to numbers.

Binary: The values of a binary attribute have only two possible values: 1 or 0, *yes* or *no*, *True* or *False*. They can also be divided into symmetric and asymmetric. In the first case, both categories have the same weight or are equally important, like gender. In the second case, both categories are not equally important, like in an outcome of a sales call,

where a client is less likely to have bought the suggested product. In chapter 5 we use this type of attribute to denote if a client made a complaint call or not.

Ordinal: The values of an ordinal attribute have a significant order between them, but there is no magnitude attributed to the values. One example of this is a scale associated with the responses in survey questions: 'like', 'neutral', 'dislike'.

Numeric: The values of a numeric attribute are quantitative and they have an order defined between them. In chapter 4 and 5 the attributes of the datasets used in the process of anomaly detection are of this kind.

2.2.1.2 Number of Attributes

Another characteristic of a dataset that can influence the choice of the anomaly detection algorithm is the number of attributes. We can have univariate datasets that only have one attribute for each of the data points or multivariate datasets that contain two or more attributes. There are specific algorithms to deal with the anomaly detection problem in both univariate and multivariate datasets, especially in a time series scenario. Despite that, multivariate algorithms can also be used for univariate anomaly detection problems, and univariate algorithms can be used for multivariate datasets performing anomaly detection individually for each attribute, not considering the relationship between them.

2.2.1.3 Relationship between Data Points

Datasets also differ in the type of relationship that the points have between each other [4]. Next, we present four different types of datasets regarding data point relations:

Tabular: The most common type is tabular data, where the data points in the dataset do not have any type of relation between themselves. In chapter 4 we test two datasets containing tabular data.

Sequential: In sequential data, like time-series, the order of each data point in the dataset matters. In these types of datasets, there is a need for modeling not only the attribute relations but also the sequential relations between consecutive data points. In chapter 5 we analyze and apply an anomaly detection approach specific for time series data.

Spatial: Spatial data is represented by numerical values in a geographic coordinate system defining the location and spatial neighborhood of the data.

Graph: In this type of data, the data instances are vertices in a graph and are linked to other vertices by edges defining a neighborhood for each data point.

2.2.1.4 Labeling of Data Points

Another important aspect is the existence of labels, binary attributes, in the dataset with the information if the data point is indeed an anomaly. Usually, in anomaly detection problems, labels are hard to obtain, since labeling is often done by human experts requiring substantial effort [4]. Because of that, all the algorithms tested are unsupervised, so

they do not need labeling to train the decision model. Nevertheless in chapter 4 and 5 we use labels containing the ground truth to evaluate the models.

2.2.2 Type of Anomalies

The three existing types of anomalies in datasets, as defined in [12, ch.12], are presented in this section.

2.2.2.1 Global

A data point is a global anomaly or point anomaly if it is significantly different from the rest of the data points in the dataset. These anomalies are the most common ones and usually the easiest to identify, and for that reason, the majority of anomaly detection algorithms were built for them. Both datasets tested in chapter 4 contain point anomalies.

2.2.2.2 Contextual

A data point is a contextual outlier if it is significantly different from the data points in the same context. In contextual anomaly detection, the context must be specified so the attributes in the dataset are divided into two groups, the contextual attributes that define the context and the behavioral attributes that define the characteristics of the data point. In this type of outlier, the same point could not be an outlier if it had occurred in a different context. It is also possible to reduce the contextual anomaly problem to a point anomaly problem since there are a lot more algorithms for point anomaly [4]. We do this in chapter 5 by applying the algorithms for point anomalies to a dataset with time-series data. Later, we compare the results of this approach with the approach where we take into consideration the context and use a time-series specific algorithm.

2.2.2.3 Collective

A subset of data points forms a collective outlier if that subset is significantly different as a whole from the rest of the dataset. Individually these points may not be outliers, but as a whole they are. A requirement for collective anomaly detection is a relationship between data points, so this type of anomaly detection is only possible in sequential and graph data. This anomaly problem can also be transformed into a point anomaly problem by transforming the sequences of points in a finite space, treating each point of the sequence as an attribute of the data point [4].

2.2.3 Output of Anomaly Detection

Anomaly Detection algorithms can report an anomaly using one of the two methods or even both of them [4]:

Scores: In this method the algorithm assigns a score to each data point, hence we can obtain a ranked list of potential anomalies, according to their degree of *outlierness*. It is

also possible to transform these anomaly scores into labels (Anomaly/Normal) using a predefined threshold on the scores. In chapter 4 all algorithms output an anomaly score, that is then transformed into labels. We use two different approaches to transform scores into labels, which we will detail in chapters 4 and 5.

Labels: Some algorithms can output directly the label of the anomaly. This is the case of the DBSCAN algorithm that groups the data into clusters, and all points that do not belong to any one of the clusters are considered an anomaly. In this case, it is not possible to rank the anomalies by their degree of *outlierness*.

2.3 Anomaly Detection Algorithms

Many different methods exist for anomaly detection. They differ in what type of outliers they detect, as well as for which type of data they are suited for. For example, algorithms used for detecting outliers in time series differ from the ones used for detecting outliers in a graph or tabular data [11]. These algorithms also differ in the type of output they provide: they can produce directly a label whether an instance is anomalous or not, or they can output an abnormality score. In this section, we are going to explain the most common algorithms used for detecting outliers. We start by defining the broad categories of algorithms that exist, and then we explain in detail the most important approaches within each category, discussing their strengths and weaknesses.

2.3.1 Categorization of Algorithms

There exist many different types of categorizations of anomaly detection algorithms. Here we present two, categorizations based on their requirement of labels or based on the approach used in detecting anomalies.

2.3.1.1 Existence of Labels

One way to categorize the algorithms of anomaly detection is according to whether the dataset has expert labeled data to build the detection model or not. In this way, we categorize the algorithms as Supervised, Unsupervised, or Semi-Supervised.

Supervised

Supervised learning requires two different types of attributes in a data set, the input attributes, and the class attribute. The purpose of this type of learning is to map the input attributes to the class attribute, thus learning the relationships between input and the class, to infer the class of new data point that is not yet classified. Hence, these supervised methods require that the data set is previously classified by specialists where each data point is either normal or anomalous. Since anomalous data is typically less frequent than the normal data, this leads to class unbalance, which might be an issue for some supervised algorithms [11].

Unsupervised

In the case of unsupervised learning, there is no labeled data. The objective is to find similar data points and group them into clusters according to their similarities. These methods assume that normal data points should be grouped, that is, they are similar between them. Normal data points do not necessarily have to be all in the same group, instead, they can be in different groups with different characteristics [12]. However, it is expected that the outliers are far from these groups of normal points. Usually, the groups of data points are calculated using distances or densities [11].

Semi-Supervised

In some scenarios, we may have a dataset in which only some of the points are classified, but most are not. For these cases, we can use semi-supervised methods. When the only labeled points are normal data points, then, we can use them together with some unlabeled points with identical characteristics to build the model, which is later used to detect outliers [12].

2.3.1.2 Classes of the Algorithms

Algorithms used for anomaly detection can be placed in five different classes [25]:

Nearest Neighbors and Clustering based methods

These classes of algorithms assume that normal points are situated closer to each other than outliers between them. Hence, they require a distance or similarity measure between data points [4].

Advantages:

- They are purely unsupervised, not making assumptions about the data distribution.
- It is easy to adapt these algorithms to all types of data by defining an appropriate distance measure.

Disadvantages:

- The techniques fail if anomaly data points are situated close to the normal ones.
- The computational complexity is high because it is necessary to calculate the distances of each data point in the testing data to all the other instances.
- In datasets with complex data, like graphs or sequences, defining the distance measures is not trivial.

Some examples of algorithms for this class are the Local Outlier Factor and the K-Nearest Neighbor Detector, which will be explained further ahead.

Classification based methods

The classification based algorithms use labels/classes in the training data to train a model that can differentiate a normal point from an anomaly one. They can be grouped into two categories depending on the number of classes: one-class if all the data points in the training data are normal or multi-class if there are normal points and anomalies [4].

Advantages:

- They can use powerful algorithms that learn functions to distinguish the data points belonging to opposing classes.
- These types of algorithms make use of a trained model so that in the testing phase each data point only has to be compared with the model, making this phase faster.

Disadvantages:

- The availability of accurate labels is usually scarce.

One example of an algorithm for this class is the One-Class SVM that will be explained further ahead.

Statistical based methods

These methods are based on the assumption that normal points are situated in high probability regions of some stochastic model and the anomalies in low probability regions as if they were produced by a different stochastic process. This class is divided into parametric and non-parametric, depending on whether they assume or not the underlying distribution of the data [4].

Advantages:

- The anomaly scores in statistical methods are associated with a confidence interval that can be used in the decision regarding the outlierness of a data point.
- If the distribution estimation is robust to anomalies, these techniques can operate without labeled data.

Disadvantages:

- The assumption that the data is generated from a particular distribution is not always true, especially in high dimensional real datasets.
- Choosing the best statistic for datasets with complex distributions is nontrivial.

One example of this class is the Gaussian Mixture Models algorithm that will be explained further ahead.

Information theoretic based methods

These techniques analyze the information content of the data using measures like entropy and relative entropy and stand on the premise that anomalies induce irregularities in the information content. The anomalies will be the minimal subset of data points that when removed from the original dataset minimize the complexity of the remainder data points [4].

Advantages:

- They can be used in an unsupervised manner.
- They do not make assumptions about the distribution of the data.

Disadvantages:

- The performance is dependant on the choice of the complexity measure.
- Using these techniques makes it difficult to attribute anomaly scores to the data points and usually, only labels are assigned.

Spectral based method

These techniques try to find a lower dimension representation for the attributes that capture the variability on the data. They assume that data can be transformed into a lower-dimensional subspace where normal data points and anomalies are notably different [4].

Advantages:

- These techniques can handle high dimensional datasets because they automatically perform dimensionality reduction.
- They can be used in an unsupervised setting.

Disadvantages:

- They are only useful if anomalies and normal data points are separable in the reduced subspace.
- Usually, these techniques have high computational complexity.

Some examples of algorithms for this class are Principal Component Analysis and Auto-Encoder, which will be explained further ahead.

2.3.2 Algorithms for Tabular Data

In practice, the unsupervised algorithms are used much more frequently than supervised, since they do not need a previously labeled data set. Besides, even with previously labeled datasets, the classes of these datasets would be very unbalanced, since the class of the anomalies would be much smaller than that of normal points and this can often interfere with the efficiency of the supervised algorithms. In this section, we will only cover the most used and most efficient unsupervised algorithms, belonging to different categories [7], [11]. The choice of the presented algorithms is also influenced by the availability of their implementation in Python. We present only a brief summary of each algorithm selected. The reader is directed to the references within the text for further details.

2.3.2.1 Isolation Forest

The Isolation Forest is an unsupervised algorithm that explicitly identifies anomalies instead of profiling normal data points. Isolation forest, like any of the other tree ensemble methods, is based on decision trees. The data set is subsampled, and an isolation tree is generated for each sample. First, an attribute is randomly selected and a split value is made by randomly selecting a value between the minimum and maximum value of the selected attribute until each data point is completely isolated from the rest. The number of partitions required, averaged over the ensemble of trees, is then used as a measure for how anomalous the data point is considered. Since anomalies are less frequent than regular observations and the values of their attributes are significantly different from the other data points, by using random partitions these anomalies are expected to be located near the root of the tree, because they are much easier to isolate. More partitions correspond to less anomalous, and fewer partitions correspond to more anomalous points. Since this type of algorithm does not use distance or density measures it has a lower computational cost: it has linear time complexity allowing the application to large datasets, although a small sample size produces better results [21].

2.3.2.2 Extended Isolation forest

The extended isolation forest algorithm is a modification of the original Isolation Forest algorithm, which consists of using hyperplanes with random slopes to perform the slicing of the data. Instead of selecting a random feature and then a random value within the range of data, it selects a random slope for the branch cut and a random intercept chosen from the range of available values from the training data. This method generalizes well into higher dimensions, where instead of straight lines it uses $N-1$ dimensional hyperplanes, where N is the number of dimensions. Extended isolation forest addressed the issue of inconsistent anomaly scores that Isolation Forest has for a certain type of data [14].

2.3.2.3 Local Outlier Factor

The Local Outlier Factor (LOF) is an unsupervised algorithm that assigns each data point a probability of being an outlier. It calculates the distance of a point to its k-th nearest neighbor and uses this to obtain a measure of density. Then the comparison is done on a local level, meaning that each data point's density is compared to its neighbors' densities, making it possible to identify regions of similar density. If a data point has a similar or higher density than its neighbors, it is considered a normal point, and if it has a lower density it is considered an outlier. The size of the neighborhood considered, the value of parameter k, has to be adapted to the dataset. A small value of k will have a much bigger local focus, performing badly if the dataset is noisy, whereas a higher value may not find local outliers [2].

2.3.2.4 Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised algorithm, widely used in dimensionality reduction, where the variables of a data set are linearly combined and reduced to its most important components. This is done by computing a linear projection of the data points into a K-dimensional subspace using Singular Value Decomposition (SVD) [29], where K is lower than the number of variables in the original dataset. The result of this calculation is a new system of coordinates composed of K principal components or eigenvectors in descending order of variance or in descending order of eigenvalues, so that the principal components with lower eigenvalues can be excluded, reducing in that way the dimensions of the original dataset [18]. Despite being widely used, the PCA has the problem of being highly influenced by the presence of noise, which increases the size of the subspace, thus making it difficult to later detect anomalies. For this reason, we have Robust Principal Component Analysis that aims to improve upon the former algorithm to be less vulnerable to noise [1].

2.3.2.5 One Class Support Vector Machine

Support Vector Machine (SVM) is a supervised classification algorithm whose goal is to find a hyperplane that separates the two different classes of the data set, and then use that hyperplane to classify new data points. The optimal hyperplane will be the one having the maximum distance from the border points from two different classes while separating them [28]. These border points are the points closest to each other, belonging to two different classes, and they are called support vectors, while the distance they have from the hyperplane is called a margin. For anomaly detection, SVM presents some problems, since we do not always have reliable labeling of the data set, and in the case that we have, the algorithm would only recognize anomalies previously existing in the training set and not the new ones [27]. For this reason, we use the One-Class SVM, the unsupervised version of SVM, that is trained only with normal data points. These points

are used to build a hyperplane, and in the classification phase, all the data points that are outside of that hyperplane are considered outliers [11]. The One-Class SVM has two important parameters:

- ν: defines the position of the hyperplane by selecting the maximum percentage of data points that could be misclassified in the training data and the minimum percentage of used support vectors.
- γ: chooses the shape of the hyperplane, where if the chosen value is too high the choice of the shape will only be influenced by the support vectors resulting in overfitting, and if it is too low the other data points will also have some influence, constraining the model.

2.3.2.6 Minimum Covariance Determinant

The Minimum Covariance Determinant (MCD) is an estimator resistant to outliers. Its objective is to find a subsample of observations whose covariance matrix has the lowest determinant. The algorithm consists of first calculating the mean of the h observations that minimize the determinant of the covariance matrix and then estimating the scatter matrix which is the covariance matrix multiplied by a consistency factor. The points that are outside of the created elliptic envelope are considered outliers, with Mahalanobis distance used to compute the outlier score [13].

2.3.2.7 Angle-Based Outlier Detection

The Angle-Based Outlier Detection (ABOD) is an anomaly detection algorithm that tries to tackle the "curse of dimensionality" by using the distances between points as a secondary criterion and using as a primary criterion the angles between the points. The angles made by outlier points in relation to the other points have less variance than the angles of normal points to each other. The Angle-Based Outlier Factor (ABOF) calculates the anomaly score of a data point as the variance of the angles that data point makes to all the remaining pairs of data points in the dataset, all this weighted by the distance of the points. Since all pairs of points in the dataset must be considered for each point, the time complexity is high. An improved algorithm called Fast Angle-Based Outlier Detection (FABOD) is used that only considers the K nearest neighbors of each point, instead of the entire dataset, where K is a parameter which should be tuned [20].

2.3.2.8 KNN Anomaly Detection

This algorithm is an adaptation of the K-Nearest Neighbor algorithm to the detection of anomalies. It starts by finding the K nearest neighbors for every point in the dataset. Then it is possible to get the anomaly score using four different ways: using the sum of the distances of the points to all its neighbors, the average of all distances to the neighbors, the median of all distances to the neighbors, or the distance to the furthest neighbor. The

choice of these four different ways to calculate the anomaly score has to be made properly for the problem in question. Also, the choice of K is very important, since using a very small or very large value could turn the density estimation unreliable [30].

2.3.2.9 Gaussian Mixture Models

The Gaussian Mixture Models (GMM) is an unsupervised learning algorithm used to discover different classes in data by computing the probability density function of data points. It is widely used in applications where it is assumed that the data comes from more than one Gaussian distribution and can be represented by the weighted sum of M component Gaussian densities. After the Gaussian mixtures for a dataset are computed a density curve is fitted and data points that have the lowest probability of belonging to these classes are classified as outliers [31].

2.3.2.10 Auto-Encoder

An Auto-Encoder is a multi-layer neural network that performs nonlinear dimensionality reduction. The number of nodes in the output layer is the same as in the input layer, and the architecture is layered and symmetric, as shown in figure 2.2. The main objective is to train the output to be as close as possible to the input. The number of nodes of the hidden layers are smaller than the nodes of the input and the output, so the only way to reproduce the input is to learn the weights so that the middle layers represent a reduced representation of the original data. The basic idea is that the outliers are much harder to be represented in the reduced representation than the normal points. Therefore, on reconstruction, an outlier will have a larger error [34].

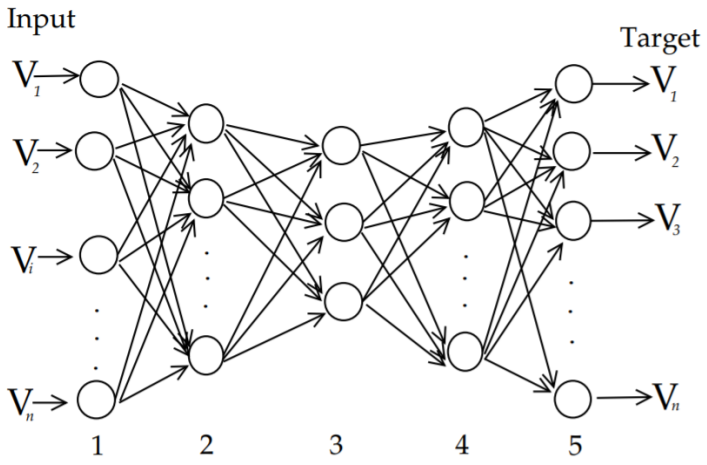


Figure 2.2: Auto-Encoder Architecture [15]

2.3.2.11 Summary of Algorithms

A summary of unsupervised algorithms presented in this section is given in table 2.1, where we show the category that each algorithm belongs to, as well as whether they should be trained only with normal points or both normal and anomaly points, even if they are unlabeled.

Table 2.1: Algorithm Overview

Algorithm	Category	Training Data
Isolation Forest	Information theoretic and Clustering	All
Extended Isolation Forest	Information theoretic and Clustering	All
Local Outlier Factor	Nearest Neighbors	Only Normal
One Class Support Vector Machine	Classification	Only Normal
Elliptic Envelope	Statistical	All
Angle-Based Outlier Detection	Nearest Neighbors	All
K-Nearest Neighbor Detector	Nearest Neighbors	All
Gaussian Mixture Models	Statistical	All
Auto-Encoder	Spectral	Only Normal
Principal Component Analysis	Spectral	Only Normal

2.3.3 Algorithms for Time Series Data

There is also a great need for anomaly detection in time-series data, which is data recorded continuously over a time period. There are two different types of time series data: univariate where only a single attribute varies over time, and multivariate which contains several attributes changing over time. Depending on the domain of anomaly detection, data types of the datasets, and the type of anomaly we want to detect, several approaches perform well in detecting anomalies. They can be out-of-limits approaches that use thresholds and raw values of the data to find anomalies, clustering-based approaches or nearest-neighbor approaches that take advantage of distance measures between data points, expert systems approaches that use rule-based methods, or dimensionality reduction approaches that try to get a lower dimensional representation of the data [17]. Recently, Recurrent Neural Networks, specifically Long Short-Term Memory (LSTM) networks, have shown very high performance when used in different anomaly detection applications with sequential data. Hence, in this section, we will only cover the LSTM Auto-Encoder architecture, which is currently considered state-of-the-art in detecting anomalies in both univariate and multivariate time-series data.

2.3.3.1 Recurrent Neural Networks

Since LSTM is a special type of recurrent neural network (RNN), we will first introduce the concept of RNN. RNN's are different from traditional feed-forward neural networks because they introduce the concept of memory, by feeding the outputs of some layers back into the inputs of a previous layer. These capabilities make these types of neural networks perfect for sequence data, such as time series, since they store and feed information from the past time steps to make predictions in the current time step, allowing them to exploit the relations between data points [33]. Although widely used, the RNN's have a flaw which is the vanishing gradient problem that prevents it from capturing the long-term dependencies in the data, making it impossible to store information for a long time [9]. To address that problem, LSTM's are explicitly designed with the capability to remember information for long periods of time.

2.3.3.2 LSTM Auto-Encoder

LSTM's were built as a mechanism to prevent the above-mentioned vanishing gradient issue in RNNs. They achieve this by having a new hidden state, called a cell. These cells have their own cell state, that function as a memory, and also gates that control the modifications made to the cell memory. Using these gates, the LSTM can know when to forget old information, when to add a new one, and what kind of information to pass as output[9]. Although the classical, fully-connected Auto-Encoder is highly effective in anomaly detection, especially for point anomalies, when we are using time-series data this method cannot capture the temporal dependencies between the data. Therefore, for detecting anomalies in time-series data, in chapter 5, we will use an LSTM Auto-Encoder that consists of a normal Auto-Encoder, where the encoder and decoder parts are built using LSTM networks instead of a regular neural network [19]. The encoder part learns a vector representation with a fixed length for the time-series input and with that representation the decoder reconstructs the time-series [24].

2.4 Performance Metrics

After we apply the algorithms for anomaly detection, we need performance metrics to assess and compare their performance. Although the algorithms are unsupervised, to test their performance we use the labels of the data points and compared them to the predictions made by the models [11]. In this section, the main metrics used in chapters 4 and 5 will be explained.

2.4.1 Fundamentals

Before we advance to the evaluation metrics, there are four terms relevant to the majority of metrics.

True Positives (TP): These are the anomalies that were correctly classified as anomalies by the algorithm.

True Negatives (TN): These are the normal points that were correctly classified as normal points by the algorithm.

False Positives (FP): These are the normal points that were incorrectly classified as anomalies by the algorithm.

False Negatives (FN): These are the anomalies that were incorrectly classified as normal points by the algorithm.

These four terms are used to construct the confusion matrix, shown in figure 2.3, which is a good visual way to understand if an algorithm is performing well. The larger the values are on the diagonal of the matrix, the better the algorithm performed.

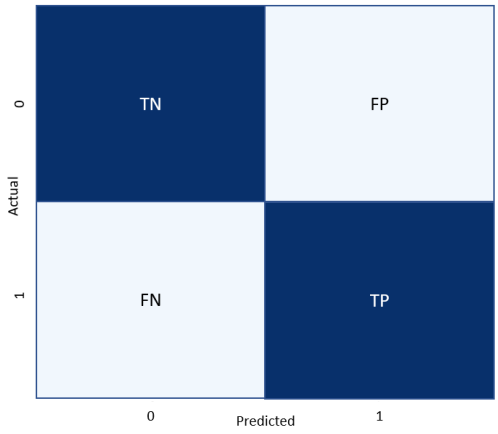


Figure 2.3: Example of a Confusion Matrix

2.4.2 Evaluation Measures

With these four terms described above, we can now approach some of the main evaluation measures: accuracy, precision, recall, and f-measure.

Accuracy is the percentage of well-classified data points in the entire test set, either normal points or outliers. It is calculated using equation 2.1, where TP and TN are the true positives and true negatives, and P and N are the total number of anomalies and the total number of normal data points, respectively.

$$accuracy = \frac{TP + TN}{P + N} \tag{2.1}$$

Precision gives us an indication of how precise is our model at predicting anomalies, that is, of all the data points that the model identified as anomalous, how many of them are anomalies. It is calculated using the equation 2.2.

$$precision = \frac{TP}{TP + FP} \tag{2.2}$$

On the other hand, **recall** tells us the percentage of true anomalies correctly classified as anomalies. That is, of all the real anomalies in the dataset, how many were identified. Recall is calculated using the equation 2.3.

$$recall = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{2.3}$$

Depending on the type of problem, we can give more importance either to precision or recall, or we can get a measure that takes into consideration both of them. This is the case of the **f-measure**. It is calculated using the equation 2.4, where β is a non-negative real number. In this dissertation we use the f-1 score to evaluate the models, where β is equal to 1, performing that way the harmonic mean of the precision and recall.

$$f_\beta = \frac{\left(1 + \beta^2\right) \times precision \times recall}{\beta^2 \times precision + recall} \tag{2.4}$$

Next, we detail some more advanced metrics, which are also used in the developed software described in chapter 3

2.4.2.1 ROC-AUC

One way to quantify the overall performance of an algorithm is by using the area under the curve of the Receiver Operating Characteristics, or ROC-AUC. The higher the value of the AUC, the better the model is at predicting the correct class, and this value ranges between 0 and 1. The ROC curve for an algorithm is plotted with the true positive rate (TPR) on the y-axis and the false positive rate (FPR) on the x-axis and this is done for all possible thresholds of the anomaly score. This metric represents the trade-off between the rate that the algorithm classifies an outlier as an outlier, and the rate the algorithm misclassifies a normal point as an outlier [12]. The equations to calculate the true positive rate and the false positive rate are in 2.5 and 2.6.

$$TPR = \frac{TP}{TP + FN} \tag{2.5}$$

$$FPR = \frac{FP}{TN + FP} \tag{2.6}$$

2.4.2.2 PR-AUC

Another important metric is the area under the Precision-Recall Curves. Here, the values of precision and recall are plotted for all possible thresholds, with the precision plotted on the y-axis and the recall on the x-axis. A perfect model is displayed as a point at coordinate (1,1) in the plot. Precision-recall curves are recommended for unbalanced datasets instead of ROC curves because the latter may show an extremely optimistic view of the performance [6].

2.4.2.3 Lift Curve

The value of the lift is calculated as follows. First, all the data points are ranked according to their value of anomaly score, from the highest to the lowest. Then we calculate the percentage of anomalies present in the top x percent of the data points and divide it by the percentage of anomalies present in the whole dataset. The value of lift at top x percent tells us how much better the algorithm is at identifying anomalies compared to a random estimator when looking at the x percent of points with the highest anomaly score. The Lift Curve is then drawn with the value of x on the x-axis and its corresponding lift value on the y-axis.

2.4.2.4 Metrics calculated without Labels

In some situations, quite common, the datasets used for anomaly detection do not have labels and for these cases, we need to use alternative metrics such as the Excess-Mass and Mass-Volume curves [10]. The intuition behind these metrics is that they try to measure how similar is the scoring function to the distribution of normal data. They rely on the fact that anomalies occur in low probability regions and use the scoring functions of the anomaly detection methods to evaluate their efficiency. They are less suited for datasets with a large number of attributes.

2.5 Data Mining Process

In the last part of this chapter, we briefly present the Cross Industry Standard Process for Data Mining (CRISP-DM) methodology illustrated in figure 2.4. The first block of this model is Business Understanding. In this phase, we try to uncover which factors can influence the results of the project. It is important to define what are the desired outputs of the project, assess the current situation by making an inventory of available resources such as the data available, and determine the data mining goals by transforming the business goals into technical data mining terms. After the block of Business Understanding comes the Data Understanding [37]. This second phase starts with initial data gathering, and consequent data description report and exploration. In this phase, we verify the data quality and obtain the first insights about the data that can help us to adapt our project plan. This is why there is a close link between Business Understanding and Data Understanding. After we have defined the business plan and identified the relevant data to our project, we start the Data Preparation phase where we get the dataset ready for modeling by cleaning, integrating, reducing, and transforming the raw data [12, ch.2]. The Data Preparation phase and the Modeling one are intrinsically linked because in the Modeling phase the objective is to select the modeling technique that best suits the problem in question. It is often in this phase that problems with data are identified or new ideas are discovered to construct new data. After the model or models have been built, we advance to the Evaluation phase where the models are validated with the metrics that

are adequate for the problem in question. In this phase, the decision is made whether the business objectives were accomplished or if changes have to be made to the Business Plan. After the model has been successfully validated, we can start the Deployment phase where the model is deployed for production.

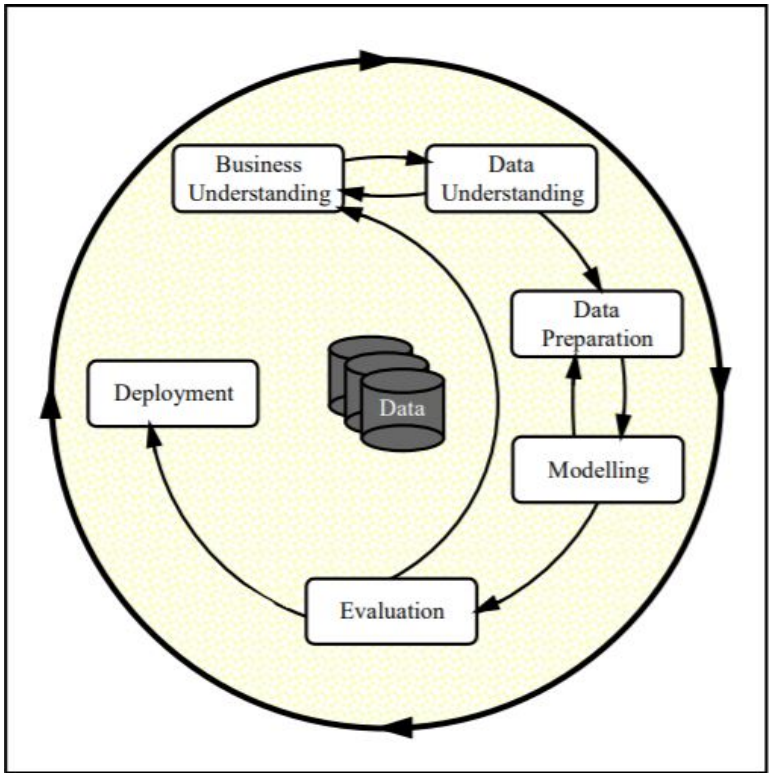


Figure 2.4: CRISP-DM Process Model for Data Mining [37]

2.5.1 Business Understanding

The business understanding is the phase where we identify the problem or the business opportunity that can benefit from a data mining approach. For example, in chapter 5 we approached the problem of the quality of service for customers of a telecommunications company and how we could identify potential malfunctions in the network that could enable the company to promptly solve those problems before the customer makes a complaint. After the identification of the business goal, the data that could potentially be used to detect these malfunctions was identified. Lastly, a project plan was made where we chose to use an anomaly detection approach for this problem and decided to evaluate this approach using information from previous customer complaints.

2.5.2 Data Understanding

The first step in data understanding is doing a descriptive statistical analysis of a dataset. This consists of describing our data using measures that can give us information about the central tendency of the data, its skewness, and the dispersion to obtain a general

understanding of the data we are working with. We can also use multiple methods for data visualization, such as histograms, scatter plots, and data correlation matrices. In this section, we describe two visualization methods used in chapter 4.

2.5.2.1 Data Visualization

Next, we describe a data correlation heatmap we used to understand the correlations between the different attributes and the labels, and a data visualization technique called t-distributed Stochastic Neighbor Embedding (t-SNE), that allows us to view high dimensional data in a 2-dimensional scatter plot.

Data Correlation: In datasets with a lot of attributes one of the best ways to check the correlation between them is by using correlation heatmaps. This is a representation of data based on the correlation matrix of the data set. We start by calculating the correlation coefficients between all the attributes, which range from -1 to 1. Attributes with a correlation coefficient of 1 are perfectly positively correlated, and with a correlation of -1 are perfectly negatively correlated. A value of 0 indicates that there is no relation between the variables. Then we create a color scale from -1 to 1 and color the matrix values accordingly.

t-SNE: This technique performs nonlinear dimensionality reduction in a dataset and is primarily used for visualizing high-dimensional data [23]. This algorithm first calculates a similarity measure between pairs of data points in the high dimensional space using a Gaussian distribution and then in the low dimensional space using a Student t-distribution. The location of points in the lower dimensional space is found by minimizing the difference between the two distributions. One important parameter of this method is the *perplexity*, which is used to manipulate the Gaussian distribution when calculating the similarity measures in the high dimensional space by influencing the variance of the distribution. This parameter will define how close similar points will be on the reduced dimension. Throughout this thesis, we will often use this visualization method to obtain insights into the performance of different anomaly detection approaches on different datasets.

2.5.3 Data Pre-Processing

Data pre-processing is the technique of transforming the raw data into data that we can understand and apply the model to. Usually, in Data mining we have large datasets, with multiple attributes, sometimes acquired from multiple sources in different formats. Therefore, before we can apply any model to our dataset, we need to pre-process our data. Data pre-processing is composed of four major steps: Data Cleaning, Data Integration, Data Reduction, and Data Transformation.

2.5.3.1 Data Cleaning

This is the step of data pre-processing that consists of assuring that the dataset does not have any values that can compromise the efficiency of the datasets. It does this by dealing with missing data and any inconsistencies in data. Although many of the algorithms already have procedures to deal with some level of noise in the dataset, they are not always completely robust, hence, this step is essential. The anomaly detection software proposed in chapter 3 has a pre-processing step to deal with missing data, by removing it or by replacing it with a different value.

2.5.3.2 Data Integration

If we want to use data from multiple sources in our analysis, these sources need to be integrated into the same dataset. To do this, we need to deal with issues such as identifying the same attributes from different databases, or different nominal values that represent the same category and cause redundancy in the system.

2.5.3.3 Data Transformation

The third step of data pre-processing involves data transformation, where data is consolidated to ensure higher efficiency of the algorithm and to obtain patterns that are easier for understanding. Data transformation consists of six steps:

- smoothing for noise removal,
- attribute construction for creating new attributes to help the mining process,
- aggregation for reducing redundant data,
- normalization for ensuring that the data falls in a smaller range (e.g. -1 to 1),
- discretization for replacing the raw values by intervals of values
- concept hierarchy generation of nominal data for substituting nominal values by values higher in the hierarchy (e.g. substituting city by the country).

The anomaly detection software proposed in chapter 3 scales the data with a standard scaler, that removes the mean and scales to unit variance.

2.5.3.4 Data Reduction

A very important step in data pre-processing is data reduction for reasons of time-saving, decreasing the computational requirements of applying an algorithm or improving the efficiency of the algorithms by removing irrelevant attributes. In dimensionality reduction, the objective is to encode the data to obtain a compressed version of the dataset. Examples of dimensional reduction are wavelet transforms and principal component analysis.

2.5.4 Modeling

In the modeling phase, appropriate algorithms are selected. The software described in chapter 3 was built to facilitate the choice of the correct anomaly detection algorithm for any type of dataset and applies ten different algorithms explained above to an arbitrary dataset, returning the performance metrics. One important characteristic of the software is its ability to perform hyperparameter tuning in each one of the algorithms to optimize them before the evaluation. A hyperparameter is a parameter of an algorithm whose value has to be chosen before the learning process begins. An example of a hyperparameter is the size of the neighborhood K in the KNN anomaly detection algorithm. Hyperparameter tuning consists of testing several parameters to assess which one is better for a specific dataset and according to the specified performance metric. In the built software, we used two different techniques of hyperparameter optimization that will be explained next.

2.5.4.1 Grid and Randomized Search

Grid search is a hyperparameter optimization technique that consists of creating a grid with ranges of different values for each one of the algorithm parameters, called the search space, and then trying all possibles combinations of parameters and evaluating which combination is the best. Randomized search is a variant of the grid search where not all possible combinations are tested. Instead, the user chooses a finite number of iterations and at each one. The combinations of parameters are randomly chosen. This variant proves to be very effective when the search space has an elevated size, achieving similar results as the standard grid search in a lot less time.

For comparing the performance of models with different parameters it is also possible to use different metrics. For example, in the software presented in chapter 3 all the algorithms were optimized using the f1-score.

Another important concept for evaluating the performance of an algorithm is K cross-validation. The dataset is divided into k equal groups and $k-1$ groups are used to train the model, while the remaining one is used to evaluate the model. Then different groups are used to train, such that after k rounds, all the groups were used to evaluate the model. In the end, the performance of the model is an average of its performances for each of the k groups. For hyperparameter tuning, each parameter combination tested is evaluated using K cross-validation.

CHAPTER 3

Description of Developed Software

The purpose of the software developed in this dissertation is to detect outliers in an arbitrary dataset using ten different anomaly detection algorithms whose parameters can be automatically optimized. The software requires two datasets as input: one with training data and the other with testing data. Providing labels is optional, meaning that the software can even be applied to datasets where no labels are available, i.e., there is no prior knowledge available on which are the normal or which are the anomalous data points. In the case where labels are available, they are used for parameter tuning and evaluation of the performance of the algorithms. Some software parameters can be manually set, such as:

- level of contamination in the dataset, i.e., the percentage of outliers in the data if it is known a priori,
- method for dealing with missing values,
- number of cross-validation folds used for parameter tuning,
- choice of classification metrics
- use of T-distributed stochastic neighbor embedding (T-SNE) visualization of the algorithm predictions.

The software returns as output:

- results of anomaly detection for each one of the algorithms,
- performance metrics (ROC-AUC, Confusion Matrix, Lift, Precision-Recall Curve, Classification Report, Mass-volume, and Excess-mass Curves),
- T-SNE visualization plot of the results,

- parameters used in each one of the algorithms.

The architecture of the software is shown in figure 3.1. The following sections detail the software parameters and the implementation of the algorithms used in each of the blocks.

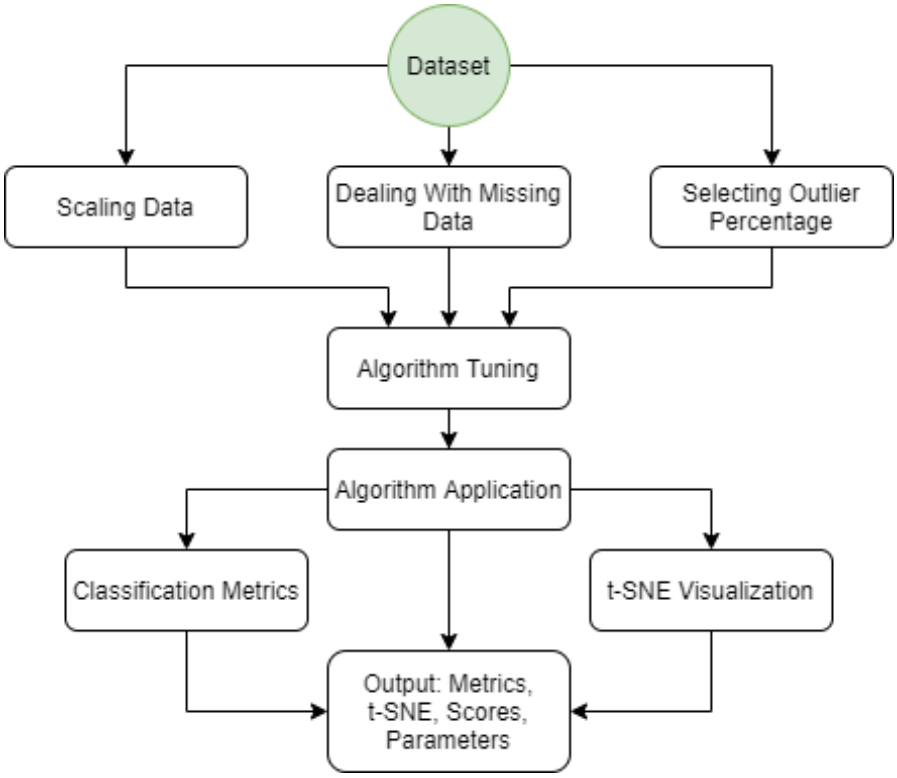


Figure 3.1: Anomaly Detection Software Architecture

3.1 Data Pre-Processing Steps

3.1.1 Dealing with Missing Data

Since most anomaly detection algorithms cannot deal with missing values in the dataset, the developed software allows the user to choose how to deal with this issue, since this is quite common in real-world datasets. We list below the implemented options for dealing with missing data.

Remove: In this option, all the instances that contain missing values are removed. This method has the disadvantage of causing some loss of information because sometimes the missing values can be an indication of an anomaly. In the case where it is known beforehand that missing values carry no additional information, this option is widely used.

Replace with chosen value: The user chooses a value that will replace all the missing values. If this value is a value not previously present in the dataset, this option allows the algorithms to learn that the case of missing data is different from all the other cases.

Replace with extreme value: If this option is selected, first the dataset is scaled using *Minmax scaler*, transforming all values on a scale of zero to one. Then, all missing values are replaced with a value of -0.1, which is a value outside the range of the scaled data. This method has the same purpose as the previous, with the advantage that it automatically finds the value not previously present in the dataset.

Replace with mean value: In this option, all the missing values are replaced with the mean of the respective numerical attribute. This method ignores the potential correlation between the features.

Replace with mean value and add an indicator for missing data: This option is very similar to the previous, except a new feature is created in a dataset. This new feature is a binary indicator, that has a value equal to one for those data points that had some value missing, and that value was imputed with a mean of the respective feature.

3.1.2 Selecting Outlier Percentage

Almost all algorithms in this software require the knowledge of the level of outliers present in the dataset. This percentage of outliers is typically referred to as the contamination level of the dataset. This value is used to transform the anomaly scores into binary labels (1 – Anomaly, 0 – Normal). The software has two options related to setting the contamination level, which are described below:

Float: The user can manually set the contamination level by inputting a number lower than 0.5.

Training: If contamination is set to "training", the value of contamination of the dataset will be calculated using the percentage of anomalies contained in the training data.

The value of contamination is then used to transform anomaly scores into binary labels, in the following way. First, all the data points whose label needs to be determined are sorted in decreasing order of their anomaly scores. Then, the instances in the top c percent of scores, where c is the chosen contamination level, are declared as anomalous, and all the others as normal. The scores are previously transformed in such a way that higher scores are attributed to instances more likely to be outliers.

3.1.3 Scaling of Data

Anomaly detection algorithms are, for the most part, based on distances between data points or on densities that depend on these distances. Therefore, the numerical values must be properly scaled, otherwise, some of the attributes could dominate others if their values are much larger than those of the other attributes. For example, the attribute that corresponds to the number of bytes downloaded by a client in one hour, if not scaled, could impact the distance much more than an attribute of the number of resets of the router done in one hour. The software performs standard scaling of each attribute, by

applying *StandardScaler* function that transforms each attribute into zero mean, unit variance variable.

3.1.4 Choosing t-SNE Perplexity

One of the outputs of the software is a t-SNE plot with the prediction results from the different algorithms. As explained in chapter 2, one of the most important parameters when calculating t-SNE lower dimension projection is perplexity. This perplexity parameter is, in a sense, a guess of how many neighbors each point has and the correct choice of this value turns the plot much more perceptible. The correct choice of this parameter will differ with different datasets, so the software has a built-in function that tests five different predefined values on the chosen dataset and outputs the plots, allowing the user to select the appropriate perplexity value when running the software.

3.2 Algorithm Implementation Details

After the pre-processing of the dataset, the software will apply ten different anomaly detection algorithms previously presented in section 2.3.2. If the user selects the option of tuning the algorithms and labels for the dataset exist, the parameters of the algorithms will be automatically tuned to find those that give the highest f1-score. The search for the optimal parameters is done by using Grid Search and Randomized Search in the training data, and by using K-Cross Validation to make the parameter tuning more robust. If the user does not select the option of algorithm tuning, the algorithms will be used with default parameter values, chosen for each algorithm accordingly. In the software, we set the algorithms to output only the anomaly scores, and the transformation to labels was done using the contamination threshold chosen in the software parameters. This way we guarantee that the transformation of the scores in labels is done the same way for every one of them, making it easy to compare the results. Below we detail the implementation of each algorithm included in the software.

3.2.1 Isolation Forest

We use the implementation of Isolation Forest from the python package *scikit-learn*. This implementation has four different parameters:

Number of estimators: Since this algorithm works as an ensemble of isolation trees, this parameter defines the number of trees in an ensemble. According to the author's original paper [21], this algorithm usually converges well with 100 estimators, so this is the value chosen as the default. However, if the tuning option is selected, the following three values are tested: 50,100,150.

Maximum samples: This parameter defines the number of samples to draw from the training data to train each one of the isolation trees. The original paper concludes that a value of 256 of maximum samples provides enough detail to perform anomaly detection,

so this is the value set as default in the software. In the case of parameter tuning, different values are tested: all powers of two, ranging from 64 to 1024. If the dataset size is smaller than the value of the maximum samples, then the value of this parameter is set to the maximum number of data points in the dataset.

Maximum features: Sometimes Isolation Forest can achieve better results if only some of the attributes of the dataset are used in the construction of each tree. The maximum features parameter controls the number of attributes used to construct each tree. By default, this value is set to use all the attributes, but if tuning is selected, the search space for the optimal parameter value is the set of five values: $0.2, 0.4, 0.6, 0.8, 1$, where these values correspond to the percentage of attributes to use.

Bootstrap: This parameter controls the sample draws in the construction of the trees. If it is set to $True$, the sampling is done with replacement, and if $False$, it is done without replacement. The default value is set to $False$.

3.2.2 Extended Isolation Forest

We use the implementation of Extended Isolation Forest available on Github [8], built using the original article [14].

This implementation has three different parameters:

Number of estimators: Since this algorithm works as an ensemble of isolation trees, this parameter defines the number of trees in an ensemble. According to the original paper, this algorithm usually converges well with 100 trees, so this is the value chosen as the default.

Maximum samples: This parameter defines the number of samples to draw from the training data to train each one of the isolation trees. The original paper concludes that a value of 256 of maximum samples provides enough detail to perform anomaly detection, so that is the value used by default.

Extension level: Since extended isolation forest uses hyperplanes (non-axis-parallel) to perform the slicing of the data, the dimensions of the hyperplane have to be chosen. This extension level defines the dimensionality of the hyperplane and its default value is set to 1. If tuning is selected, the search space for the optimal value of the hyperplane dimension is a range of values from 1 to $n-1$ where n is the number of features in the dataset.

3.2.3 Local Outlier Factor

We use the implementation of Local Outlier Factor from the python package *scikit-learn*.

This implementation has three different parameters:

Number of neighbors: This parameter defines the number of nearest neighbors used to measure the local density for each point. The default value chosen for this parameter is 20, but if tuning is performed, the search space is a list of values ranging from 10 up to one-third of the dataset size.

Metrics: This algorithm is distance-based, and therefore it is necessary to choose the metric for the distance calculation. The default value was set to the Euclidean distance since it is one of the most used metrics in distance-based algorithms. If algorithm tuning is to be performed, then the following four metrics are tested and compared: Euclidean, City-block, Cosine, and Chebyshev.

Algorithm: This implementation of the Local Outlier factor allows for the selection of the algorithm responsible for computing the nearest neighbors. There are three possible algorithms: Ball tree, KD tree, and Brute-force search. The default value for this parameter is the 'auto' and in this option, the algorithm will be chosen based on the other input values.

After parameter selection, the Local Outlier Factor is trained using only the normal data points in the training data if labels are available and then tested on the testing data.

3.2.4 One Class Support Vector Machine

We use the implementation of the One-Class Support Vector Machine from the python package *scikit-learn*.

This implementation has three main parameters:

Kernel: This parameter defines the function used for building the hyperplane that separates the normal data from the outliers. The default kernel used in the software is Radial Basis Function (RBF) because it showed superior performance in multiple tests we performed with the software.

ν: This parameter is bounded between 0 and 1, and represents an upper bound on the fraction of margin errors and a lower bound on the fraction of support vectors. Its value should be equal to the proportion of outliers we expect to observe in the dataset. The default value is set to the value of contamination in the dataset and when tuning is performed, this parameter is selected from a list of values starting from 0.01 and going up to 0.5 with steps of 0.01.

γ: This parameter corresponds to the kernel coefficient for the RBF kernel. If this parameter is too high, the area of influence of the support vectors, which represents the area of data points that influence the choice of the hyperplane, only contains the support vector itself resulting in overfitting. If the parameter is too low, the model will be too constrained and will not be able to capture the shape of the data. The default value for this parameter is set to 'scale' which corresponds to $1/(nFeatures * X.var())$ where $nFeatures$ is the number of features in the dataset and $X.var()$ is the variance computed on the flattened dataset, that is a dataset that was transformed in a one-dimensional array. If the parameter tuning is enabled, this value is selected from values spaced evenly on a log scale starting with 10^{-9} and going to 1000.

After parameter selection, One-Class SVM is trained using only the normal data points in the training data if labels are available and then tested on the testing data.

3.2.5 Elliptic Envelope

We use the implementation of Elliptic Envelope from the python package *scikit-learn*.

This implementation has two main parameters:

Assume Centered: This is a boolean parameter that, if set to $True$, the algorithm estimates the covariance of the dataset without centering the data. This option is very useful to use on datasets with a mean very close to zero. When set to *False*, the robust location, that represents the central location of the dataset if the dataset did not have any outliers, and the covariance are estimated using the FastMCD algorithm [32] directly. The default value of this parameter is *False* and when tuning is enabled, both $True$ and *False* options are tested.

Support Fraction: This parameter is a value between 0 and 1 and it corresponds to the proportion of points in the training data that should be included for calculation of raw minimum covariance determinant (MCD). The default value is calculated using $[nSample+nFeatures+1]/2$, where $nSample$ is the total number of samples in the training data, and $nFeatures$ is the total number of features. This default value represents the actual number of points to use for the calculation and not the percentage. If tuning is enabled, the parameter is selected from the list of values starting from 0.1 up to 1, with increments of 0.1, containing also the value $[nSample+nFeatures+1]/2$.

3.2.6 Fast Angle-Based Outlier Detection

We use the implementation of Angle-Based Outlier Detection from the python package *pyod* [38].

This implementation has two main parameters:

Method: This parameter has two options, the 'default' where the algorithm trains the model using all the training points and the 'fast' where the algorithm trains the model using only the nearest neighbors of a training point. In our software, we only used the 'fast' option, because, with the other option, the execution time of the algorithm for large datasets was too long.

Number of neighbors: This parameter defines the number of nearest neighbors used when the 'fast' method is selected. The default value chosen for this parameter was set to 10, but in the case of algorithm tuning, the value is selected from a list starting from 10 and going up to 90, with increments of 20.

3.2.7 K-Nearest Neighbor

We use the implementation of K-Nearest Neighbor from the python package *pyod* [38].

This implementation has three main parameters:

Number of neighbors: This parameter defines the number of nearest neighbors to use when calculating the anomaly scores. The default value chosen for this parameter

was set to 20, but in the case of algorithm tuning, the value is selected from a list starting from 10 up to one-third of the number of data points.

Method: This parameter has 3 options, 'largest' where the algorithm uses the distance to the furthest neighbor as the outlier score, 'mean' where the algorithm uses the average of the distance to all the neighbors as the outlier score and the 'median' option that uses the median of the distances to all neighbors. As the default option, the software uses the 'largest' option, and when the tuning is selected, all three options are tested.

Metrics: This algorithm is distance-based, and therefore it is necessary to choose the metric for the distance calculation. The default value was set to Euclidean distance since it is one of the most used metrics in distance-based algorithms. If algorithm tuning is to be performed, then the following three metrics are tested and compared: Euclidean, City-block, and Chebyshev.

3.2.8 Gaussian Mixture Models

We use the implementation of Gaussian Mixture Models from the python package *scikit-learn*.

This implementation has three main parameters:

Number of Components: This parameter defines the number of Gaussian distributions used to fit the data. The default value is 5, but when tuning is selected the value is chosen from a list starting from 1 up to 10.

Covariance Type: This algorithm provides four different options to calculate the covariance: 'full' where each component has its own general covariance matrix, 'tied' where all components share the same general covariance matrix, 'diag' where each component has its own diagonal covariance matrix, and 'spherical' where each component has its own single variance. By default, this parameter has the value 'full', and when parameter tuning is selected, all four options are tested.

Parameter Initialization: The weights, means, and precisions of the algorithm have to be initialized with some values in the training process and for that, the algorithm has two different options, 'kmeans' where the initial parameters are initialized using the algorithm K-means and 'random' where the parameters are initialized randomly. By default, this parameter has the value 'kmeans' option, and when parameter tuning is selected, both options are tested.

3.2.9 Auto-Encoder

We use the implementation of Auto-encoder from the python package *pyod* [38].

The chosen architecture for the auto-encoder hidden layers is calculated based on the number of attributes in the dataset so that the dimensionality reduction, discussed in section 2.3.2.10 is well performed.

This implementation has three main parameters:

Hidden Activation: This parameter defines the activation function used in the hidden layers. The default activation function is the Rectified Linear Unit (ReLU) but when tuning is selected, five different activation functions are tested: ReLU, Hyperbolic Tangent (TanH), Scaled Exponential Linear Unit (SELU), SoftMax and Sigmoid.

Output Activation: This parameter defines the activation function used in the output layer of the auto-encoder. The default activation function is Sigmoid, but when tuning is performed, six different activation functions are tested: ReLU, TanH, SELU, SoftMax, Sigmoid, and Exponential.

Dropout: The Dropout parameter defines the rate of dropout inside the autoencoder, i.e., the percentage of neurons that are randomly disabled during training to avoid overfitting of the model. The value of this parameter is set to 0.2.

3.2.10 Principal Component Analysis

We use the implementation of Principal Component Analysis from the python package *pyod* [38].

This implementation has four main parameters:

Number of Components: This parameter defines the number of principal components used to reduce the dimensionality of the dataset. The default value used is 'mle', which defines that the ideal number of principal components is calculated using Minka's maximum likelihood estimation (MLE) [26]. This value is not tuned.

SVD Solver: This parameter defines the method used to perform Singular value Decomposition (SVD). By default, this parameter is set to 'full', which combined with the value 'mle' of the parameter Number of Components defines that the ideal number of principal components is calculated using Minka's maximum likelihood estimation (MLE) [26]. This value is also not tuned.

Whiten: When this parameter is set to $True$ the vectors containing the components are multiplied by the square root of the number of samples in the dataset and then divided by the singular values to ensure uncorrelated outputs. By default, this parameter is set to $False$, but when tuning is performed, both values, $True$ and $False$ are tested.

Weighted: When this parameter is set to $True$, the eigenvalues are used to compute the anomaly scores by adding the weighted distances between each sample to the eigenvectors. The eigenvectors with smaller variance come with more importance in the outlier score calculation [38]. By default, this parameter is set to $True$, but when tuning is performed, both values, $True$ and $False$ are tested.

Chapter 4

Comparison of Anomaly Detection Algorithms

We used the developed software, described previously in Chapter 3, to compare the performance of different anomaly detection methods for solving two different real-world problems. In the first case, we detect fraudulent credit card transactions, and in the second case, we identify telecom users who will miss their payment, i.e., perform involuntary churn. For tackling the problem of credit card fraud, we use a publicly available Kaggle [5] dataset, and for addressing the problem of involuntary churn, we use a client payment dataset from a telecom company. In the following sections, we will provide more details about the datasets, as well as describe the performance of different algorithms used to address these two problems.

4.1 Detecting Credit Card Fraud

The dataset we used contains information about credit card transactions that occurred during two days in September of 2013. The original dataset has 284,807 transactions, with 492 of these transactions being labeled as fraudulent. Each transaction is characterized by 30 features, with 28 of them being the result of a PCA transformation and represent the projection of original data on obtained principal components. The remaining 2 features are the amount of money in the transaction and the time elapsed between the respective transaction and the first transaction in the dataset. Each transaction also contains a label that is set to 1 if the transaction is fraudulent, otherwise, it is 0. We will be referring to instances with label 1, as anomalous class, and those with label 0 as normal class.

4.1.1 Data Pre-Processing

Sub-Sampling: We sampled the number of instances to reduce the running time of algorithms. We selected all of the 492 fraud transactions from the dataset and 10,000 random normal transactions, obtaining in that way a percentage of anomalies of 4.7%.

Train/Test Split: The resultant subset was split into two equal parts, with the first half used for training, and the second for testing.

Feature Sub-Sampling: Since some of the algorithms we wanted to test take a prohibitively long time to process high dimensional data sets, specifically Fast Angle-Based Outlier Detection, we reduced the number of features in the dataset. This reduction of the number of features was also implemented to allow us to test the performance metrics of the excess-mass and mass-volume curves since they typically have inferior performance for datasets with high dimensions[10]. Since the features present in the dataset are already output of PCA, we selected those features which are more correlated with the class. The chosen features were $V14, V12, V17$, which are the most negative correlated features, and $V11, V4, V2$, which are the most positive correlated features, as illustrated in figure 4.1.

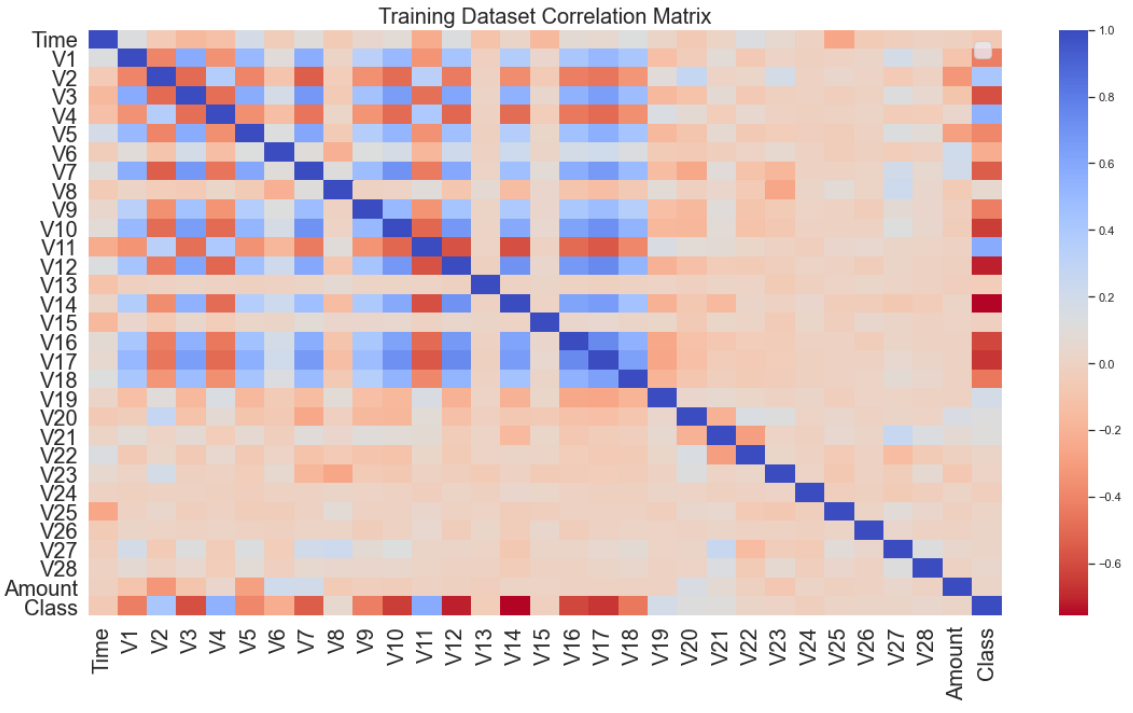


Figure 4.1: Correlation Matrix from credit card fraud training dataset

We summarize the most important aspects of the dataset in table 4.1, indicating the total number of data points and features, the number of samples and features containing missing values (NA), the level of anomaly contamination in the dataset, and the train/test split percentages.

Table 4.1: Credit Card Dataset Overview

Number of Samples	Samples with NA	Number of Features	Features with NA	Contamination [%]	Train/Test Split [%]
10492	0	6	0	4.7	50/50

4.1.2 Data Visualization

To gain further insight into the complexity of the problem, we use the t-SNE approach to obtain a two-dimensional representation of the test dataset, shown in figure 4.2. The t-SNE dimensionality reduction was applied with the perplexity parameter as 70, because with this value we got the best separation between fraudulent and normal data points in the testing dataset. The orange dots represent the fraudulent transactions and the blue dots represent the normal ones. We can observe that most of the fraudulent transactions form a cluster at the top of the plot, meaning that the majority of fraudulent transactions are similar to each other and different from the normal transactions.

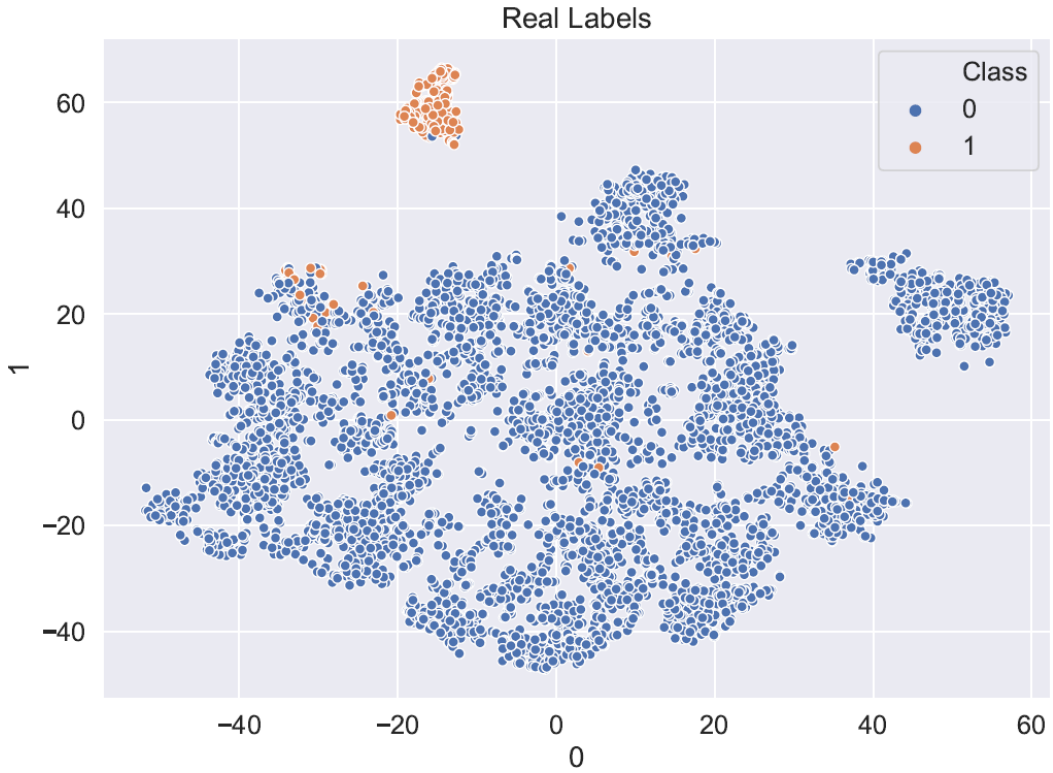


Figure 4.2: Credit card fraud t-SNE with real labels

4.1.3 Results

The software described in the previous chapter was used to identify anomalies in this dataset with the parameter tuning enabled and disabled for comparison. The anomaly contamination parameter of the software, that is used to transform the anomaly scores of

the testing data into labels, as described in section 3.1.2, was selected as the percentage of anomalies in the training dataset.

Since we randomly split the dataset into training and testing, the anomaly percentage in the testing subset was approximately equal to the anomaly percentage observed in the training subset.

As mentioned in the previous section the dataset does not contain any missing data, therefore, the only pre-processing performed by the software was the standard scaling, where the features were scaled to have a mean of 0 and a variance equal to 1.

When hyperparameter tuning was performed, the number of K cross-validation folds was set to two. We chose a low value of K as a trade-off between the robustness of hyperparameter tuning and the time consumption. Even though a higher value of k could lead to a more robust choice of parameters, it was prohibitively costly in terms of time.

4.1.3.1 Tuning Gain

After the results with and without tuning are obtained we analyze the advantage of parameter tuning in the anomaly detection algorithms by plotting the resultant f1-Score of both cases. In figure 4.3, we can observe that only three of the algorithms (Fast Angle-Based Outlier Detection, K-Nearest Neighbor Detector, Gaussian Mixture Models) have significant gains in the f1-score in the tuned version. There are multiple explanations for the low gains in the f1-score values of the majority of the algorithms:

- default values of the algorithm´s parameters achieve already good results in this dataset,
- low dimensionality of this dataset reduces the importance of a large variation in the parameters to obtain good results,
- difficulty of the dataset is relatively low, as illustrated in 4.2, since the anomalous data points are similar to each other and much different from the normal ones.

Next, we discuss in detail the three algorithms with significant f1-score gains.

Fast ABOD

For the fast ABOD only one parameter was optimized and that is the number of neighbors used to calculate the anomaly score. We only searched for the best parameter over a small range of values, since this algorithm has a high running time, with complexity $O(n^2+n\times k^2)$, where n is the number of data points in the testing data and k is the number of nearest neighbors we are using. Since the running time increases with the size of neighbors used we could not use high values of neighbors for this algorithm, and this may explain its low f1-score and also its relatively high gain when raising the number of neighbors. We can observe in table 4.2 that choosing a higher number of neighbors leads to the improvement of f1.

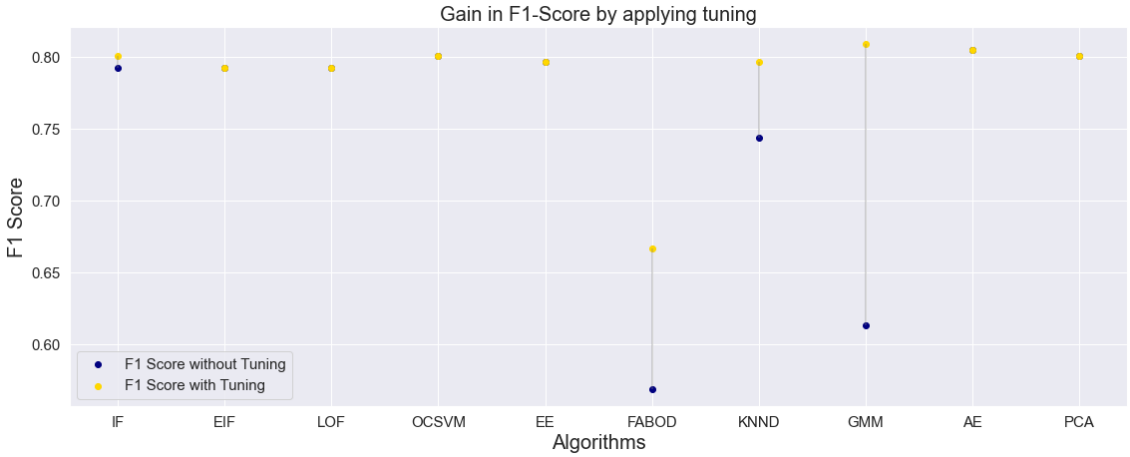


Figure 4.3: Credit card fraud tuning gain

Probably, if we further increased the number of neighbors, we might have obtained even better performance, but this was not computationally possible.

Table 4.2: Fast ABOD default and tuned parameters in credit card fraud dataset

Parameters	Contamination	Method	Number of Neighbors
Default	0.05	Fast	10
Tuned	0.05	Fast	90

K-Nearest Neighbor Detector

For this approach, raising the number of neighbors and changing the metric for distance calculation improved the f1-score. As we previously mentioned, the KNN Detector algorithm chooses as anomalous points those whose distance to the furthest k-th neighbor is the largest. In this case, anomalous points, as illustrated by the figure 4.2, are clustered together and not scattered, hence with a low number of neighbors, the furthest near neighbor of an anomalous data point will be another anomalous data point, relatively close, resulting in a low anomaly score. The parameters used in the default and tuned cases are given in the table 4.3.

Table 4.3: K-Nearest Neighbor Detector default and tuned parameters in credit card fraud dataset

Parameters	Contamination	Number of Neighbors	Method	Metric
Default	0.05	20	Largest	Euclidean
Tuned	0.05	92	Largest	Cityblock

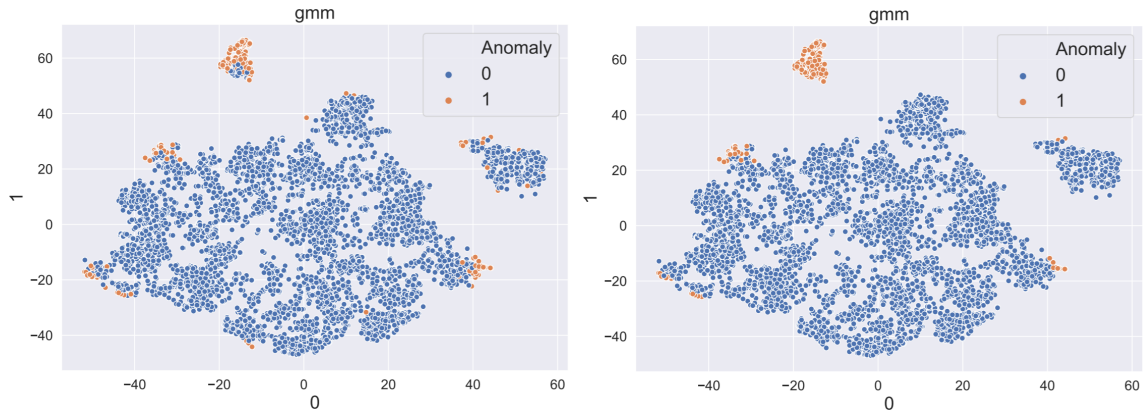
Gaussian Mixture Models

In this algorithm, three of the parameters were changed in the tuned version, explaining the gain in f1-score. One of the parameters was the number of Gaussian components which changed from 5 to 6, meaning that this dataset has a higher value of different distributions than the default value. Also, the covariance type and the parameter initialization method were changed. The table 4.4 contains the parameters used in the default and tuned cases.

Table 4.4: Gaussian Mixture Models default and tuned parameters in credit card fraud dataset

Parameters	Contamination	Nr. of Components	Covariance	Initialization
Default	0.05	5	Full	K-means
Tuned	0.05	6	Diag	Random

Figure 4.4 showcases the tuning gain, with two t-SNE plots, the first one showing the labels predicted by the Gaussian Mixture Model algorithm with the default parameters, and in the second plot showing the labels predicted by the tuned model. These predicted labels can be compared with the real labels in figure 4.2. We can see that the model whose parameters have been tuned has a fewer number of false positives, as well as false negatives.



a t-SNE with predictions without tuning

b t-SNE with predictions with tuning

Figure 4.4: Comparison of Gaussian Mixture Models predictions between the default and tuned versions in credit card fraud dataset

4.1.3.2 Performance Metrics

In table 4.5 we summarize the performance of different anomaly detection algorithms applied on the dataset of credit card fraud. Further below, we will discuss different metrics of performance.

Time

The column of time in table 4.5 corresponds to the time in seconds that each algorithm took for tuning, training, and testing. The differences in the magnitude of the values for the time each algorithm took to run can be explained by these factors:

- time complexity of the algorithm.
- size of the search space in the hyperparameter tuning.
- implementation of the algorithm in Python, since some packages are more optimized than others.

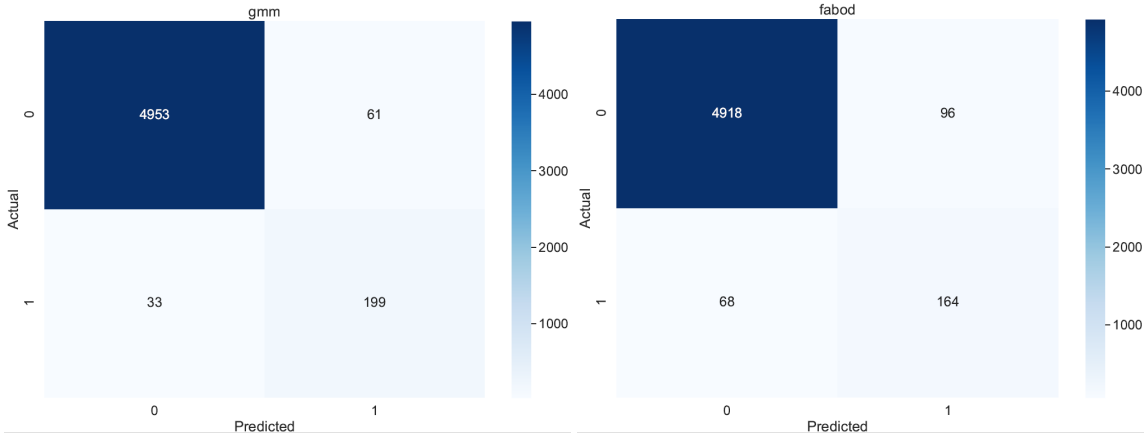
This being said, two algorithms took a much longer time than the remainder. The Auto-Encoder has an execution time almost six times higher than the third slowest algorithm. This happened because every time the tuning version tries one different activation function it has to retrain the neural network which takes time. Then we have the Fast ABOD algorithm with a time value almost thirty-four times higher than the third slower algorithm because even though only one parameter is tuned and the parameter search space is relatively small, the time that this algorithm takes to run is quite high and increases with the increasing of the number of neighbors as explained before.

F1-Score, Recall and Precision

The algorithm with the best f1-score is the Gaussian Mixture Model with a score of 0.809, and it is also the algorithm with the best Recall and Precision. Figure 4.5 shows the confusion matrix of the algorithm with the highest score, Gaussian Mixture Model, and the one with the lowest score, Fast ABOD. We can observe that the GMM model identified a higher number of fraudulent transactions, but also, it made fewer errors of labeling fraudulent transactions as normal and vice versa. We additionally illustrate this difference in figure 4.6 with two t-SNE plots, side by side, of the predictions of each of these algorithms. We can observe that the Fast ABOD does not detect all the anomalies in the cluster at the top, and also it has a higher number of false positives scattered in other areas of the dataset. In appendix A we present the t-SNE plots with the results from all the other algorithms applied to this problem that were not demonstrated here.

Table 4.5: Performance of different anomaly detection algorithms for the credit card fraud dataset

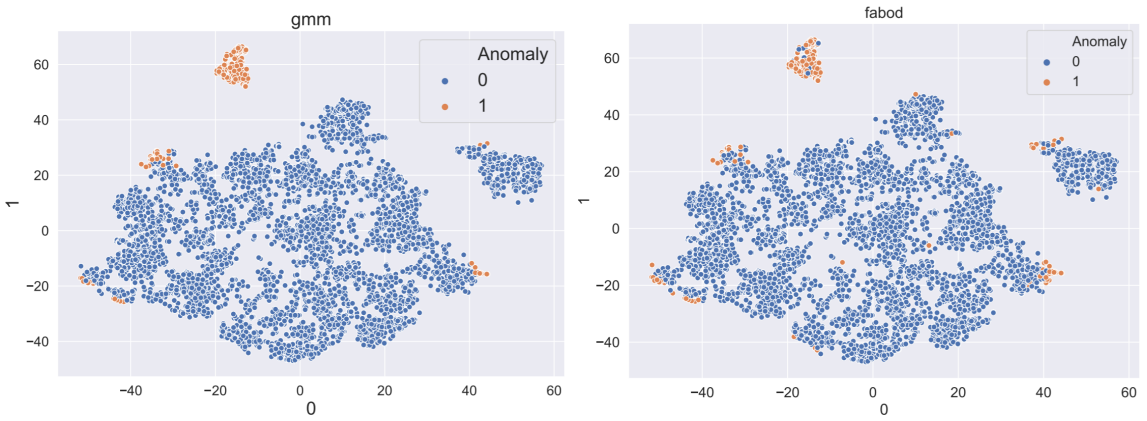
Algorithm	Time	Lift_1	Lift_10	Roc_Auc	PR_Auc	F1	Recall	Precision	Accuracy	EM_Auc	MV_Auc
isolation_forest	89	22.61	9.02	0.966	0.879	0.801	0.849	0.758	0.981	0.00364	82503
ext_isolation_forest	285	21.31	8.93	0.964	0.838	0.793	0.841	0.750	0.981	0.00381	40991
local_outlier_factor	68	22.61	8.89	0.956	0.861	0.785	0.832	0.742	0.980	0.00341	39874
one_class_SVM	277	22.61	8.72	0.945	0.857	0.801	0.849	0.758	0.981	0.00256	47585
elliptic_envelope	55	22.18	8.93	0.966	0.871	0.797	0.845	0.754	0.981	0.00378	70441
fabod	9608	16.96	8.93	0.961	0.616	0.667	0.707	0.631	0.969	0.00395	559
knnd	226	21.31	9.11	0.968	0.813	0.797	0.845	0.754	0.981	0.00378	14273
gmm	8	14.78	9.02	0.961	0.628	0.809	0.858	0.765	0.982	0.00216	232393
auto_encoder	1589	21.74	9.02	0.966	0.869	0.805	0.853	0.762	0.982	0.0036	48900
pca	1	21.74	9.02	0.964	0.867	0.801	0.849	0.758	0.981	0.00357	49376



a Gaussian Mixture Model Confusion Matrix

b Fast ABOD Confusion Matrix

Figure 4.5: Comparison between the confusion matrix of Gaussian Mixture Models and Fast ABOD in the credit card dataset



a Gaussian Mixture Model t-SNE plot

b Fast ABOD t-SNE plot

Figure 4.6: Comparison between the t-SNE plots of Gaussian Mixture Models and Fast ABOD for the credit card dataset

ROC-AUC and PR-AUC

Next, we compare the performance of the algorithms by looking at the values of the area under the ROC (ROC-AUC) and Precision-Recall curves (PR-AUC). In terms of the area under the ROC Curve, all the algorithms have similar performance as we can see in figure 4.7. Similar conclusions are drawn by looking at the values of the area under the Precision-Recall curve from the values in the table 4.5. For both curves, Fast ABOD has the worst performance, for the reasons previously explained. Isolation forest has the highest PR-AUC, while KNND has the highest ROC-AUC, though very similar to Isolation Forest. It is interesting to observe that GMM, a model with the highest precision and recall has quite low PR-AUC, and not the highest ROC-AUC. The area under the curves assesses the performance of the models for all the possible thresholds, while precision

and recall are for a fixed threshold. Hence, we can state that, while for the threshold we used, based on true contamination level, GMM does a relatively good job, this does not hold for all the other threshold values, as we can see in figure 4.8. Based on table 4.5 the threshold chosen for the GMM is where the recall is around 0.86 and the precision is around 0.76 and that it is the exact spot where the precision-recall curve for the GMM overtakes the other curves in figure 4.8.

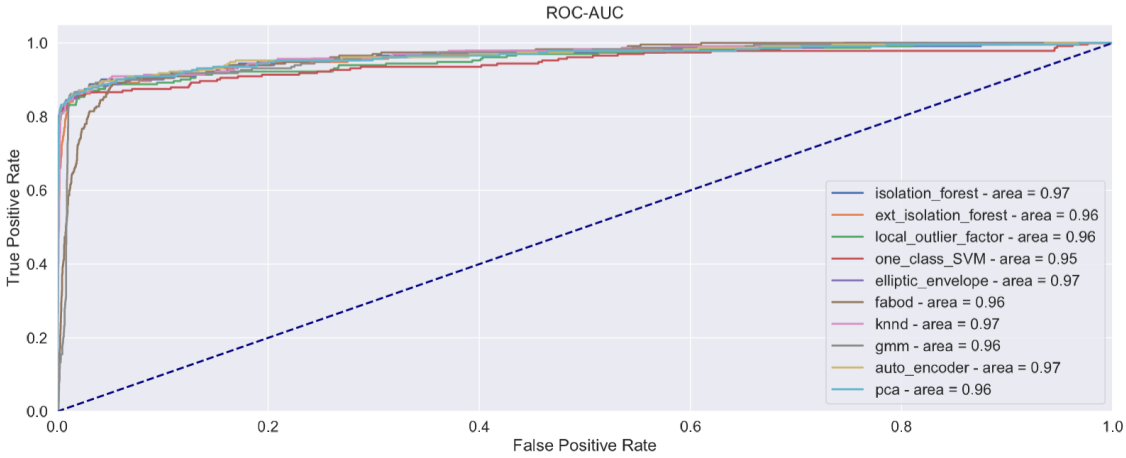


Figure 4.7: Credit card fraud ROC Curves

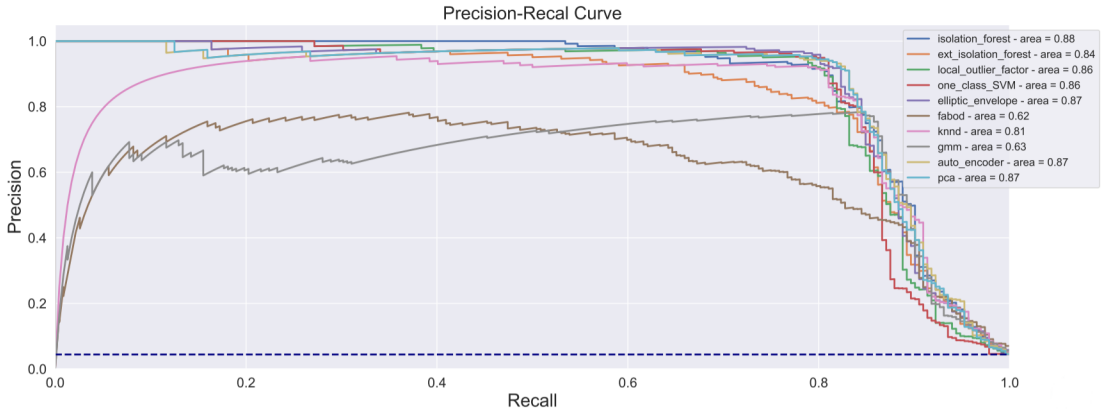


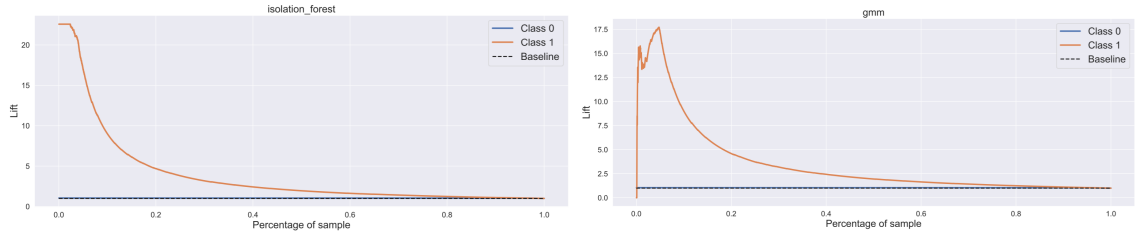
Figure 4.8: Credit card fraud PR Curves

Lifts

Next, we discuss the lift for the first percentile and the first decile, again looking at the values in the table 4.5. As expected, the gain in the one percent of the test set is much higher than in the rest of the dataset, with the lift being around 20 for most of the algorithms. This means that if we randomly selected 1% of the test dataset we will have 52 data points, out of which 2 are anomalous. However, if we look at the 1% data points with the highest anomaly score we end up having around 40 anomalous data points out of the 52 data points, 20 times higher than with random selection. Two of the algorithms with the smaller lifts in the first percentile are the Fast ABOD and the Gaussian Mixture

Models, and this happens because the points that these algorithms consider as the most anomalous are not anomalies at all, meaning that different algorithms consider different characteristics as an anomaly indicator, as illustrated earlier in the figure 4.6. Although in the first percentile these algorithms have the worst results, when looking in the first decile they have an equivalent performance to the other algorithms. As mentioned earlier, the GMM algorithm has actually the best results in terms of precision and recall for the set contamination level. Since the anomaly contamination in the dataset was set to approximately 4.7%, which is nearly half of the first decile, this is consistent with GMM algorithm having an overall good performance in the first decile.

In figure 4.9 we can see the difference in lifts between the Isolation forest, that has good results for the lifts in both the first percentile and the first decile, and the lifts of Gaussian Mixture Models that only has good results in the first decile. By comparing the t-SNE plots of Isolation Forest and the Gaussian Mixture Models predictions in the first percentile, we can infer why the Isolation forest has superior values of lift for the first percentile. As we can see in figure 4.10, the Isolation forest predicts anomalies only in the top cluster where the majority of real anomalies are situated, as we can see in figure 4.2. However, GMM, although also finding a good part of anomalies in the top cluster, it also falsely identifies some scattered normal points as anomalous.



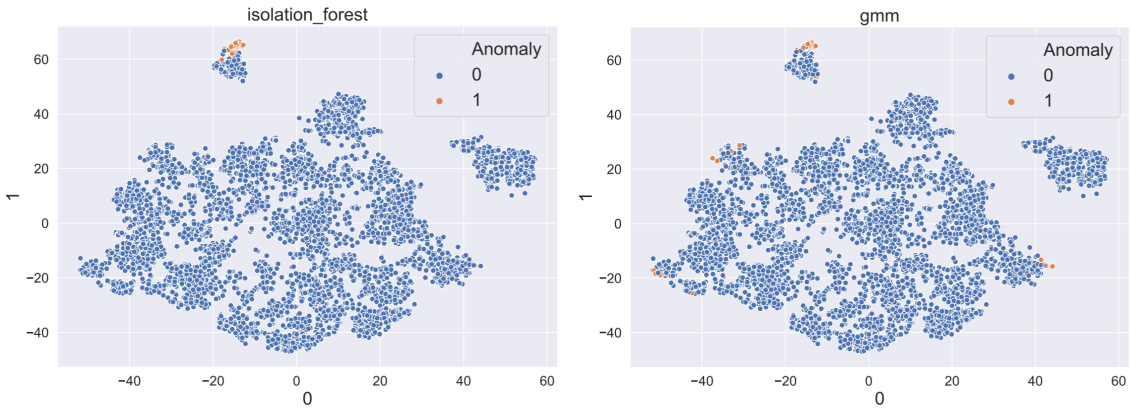
a Isolation Forest Lift

b Gaussian Mixture Models Lift

Figure 4.9: Comparison of lift plots between the Isolation Forest and Gaussian Mixture Models predictions in the credit card dataset

Excess-Mass and Mass Volume

We also analyze the results from the metrics used to compare the performance of different algorithms when labels are not present, EM and MV, presented in Section 2.4.2.4. Given that for this dataset we also know the true labels, it allows us, not only to compare performances of different algorithms using these new additional metrics, but also to compare these metrics to more traditional ones, like ROC-AUC and PR-AUC, which were previously discussed. What we can note by looking at the values again in the table 4.5, is that the ranking obtained with these metrics does not match the ranking obtained by using the previously discussed metrics. Note that the higher values of area under the curve of the Excess-Mass curve are the best results, while in the Mass-Volume curve, the best results are the smallest values. The algorithm with the best results in these metrics is the algorithm that has the worst results in the metrics that use ground truth labels. The



a Isolation Forest t-SNE plot

b Gaussian Mixture Models t-SNE plot

Figure 4.10: Comparison of t-SNE plots between the Isolation Forest and Gaussian Mixture Models predictions in the first percentile on the credit card dataset

result, although not intuitive, is not surprising. As explained by the authors [10], this may happen when no algorithm is a clear winner according to both metrics ROC-AUC and PR-AUC, which is exactly our case. The authors note that they also observe that only when one algorithm beats the others at both metrics, it also has matching ranking obtained by EM and MV. Both of these metrics, EM and MV, evaluate the performance of the algorithms by measuring their ability to detect points in low probability regions, which is actually what the Fast Angle-Based Outlier Detection does, hence it achieves the best score according to these two metrics. This also happens with the other two algorithms that are based on distances, K-Nearest Neighbor Detector and the Local Outlier Factor, which have the second and third best Mass-Volume results. Although these methods can be used to compare algorithm performance when we do not have labels, still, when actual labels are present, other metrics might give a more complete picture.

4.2 Detecting Involuntary Churners

Another problem approached in this dissertation was the detection of involuntary churners. A churner is a customer, in this case from a telecommunications company, that ends his contract with the company willingly. There are also involuntary churners that are the customers whose contract is terminated due to their lack of payment. This dataset contains information about the payment of bills for a sample of customers of a telecommunications company. Every instance of the dataset represents one customer at a specific time with the information whether the customer paid before the deadline or did not pay at all and thus, consequently churned. The dataset has $10,000$ customers that paid before the deadline and $1,500$ that did not pay. We formulate this as an anomaly detection problem by labeling those $1,500$ churners as anomalies. Each client is characterized with nine features:

- five represent the amount of the invoices from the last five months,
- two represent the quantity of KB downloaded in the last thirty and ninety days for fixed Internet service they contracted,
- two represent the quantity of KB uploaded in the last thirty and ninety days.

Each client also has a label, with the value 1 if the client churned and 0 if they paid before the deadline.

4.2.1 Data Pre-Processing

The only pre-processing used before the application of the software was the split in train and test datasets.

Train/Test Split: We randomly split the dataset into two equal subsets: training and testing dataset.

Also, 20% of this dataset has missing values in 4 of the features, that are the features related to the quantity of downloads and upload. This represents customers that do not have Internet by fiber optic. We summarize the most important aspects of the dataset in table 4.6, indicating the total number of data points and features, the number of samples and features containing missing values (NA), the level of anomaly contamination in the dataset, and the train/test split percentages.

Table 4.6: Involuntary Churn Dataset Overview

Number of Samples	Samples with NA	Number of Features	Features with NA	Contamination [%]	Train/Test Split [%]
11500	3237	9	4	13	50/50

4.2.2 Data Visualization

To gain further insight into the complexity of the problem, we once again use the t-SNE approach with the perplexity parameter at 70 to obtain a two-dimensional representation of the test dataset. In figure 4.11a we can observe by looking at the distribution of the anomalies (orange dots) in the t-SNE plot, that this dataset seems more complex since the anomaly points, although forming some groups, are a lot more spread out and mixed with the normal points. Unlike in the previous case, here the anomalous points, i.e., client churners, no longer form a distinct cluster. This was expected, since clients may stop their payment due to many different reasons, and many of these reasons are not captured by the variables we have present in the dataset: Internet usage and prior invoices. However, we still applied different anomaly detection algorithms to this dataset to assess and compare algorithm performance in a challenging real-world context.

In order to gain better understanding of the problem and compare the performances of different algorithms, we analyzed the characteristics of the customers from our dataset. Since the t-SNE plot showed that the customers formed five distinct clusters, we analyzed each one of them in detail. With each of these clusters colored differently in figure 4.11b, we list below the characteristics of each group of customers:

- Cluster 1 (Blue): Contains the customers that do not have Internet by optic fiber and the customers that have Internet by optic fiber but their monthly download and upload are 0 or values very close to 0. Another important observation is that the scattered points inside the circular cluster represent customers that have some of the invoices amounts equal to 0.

- Cluster 2 (Orange): Contains customers that have Internet by optic fiber but their monthly values of download and upload are low, in the hundreds of KB, but still higher than 0. These are also customers that have a higher quantity of uploads than downloads. Also, the values of the invoices are low when compared with the Clusters 1 and 5.

- Cluster 3 (Green): Contains customers that have Internet by optic fiber and their monthly values of download and upload are low, with values in the order of thousands of KB. Also, the values of the invoices are low when compared with the Clusters 1 and 5.

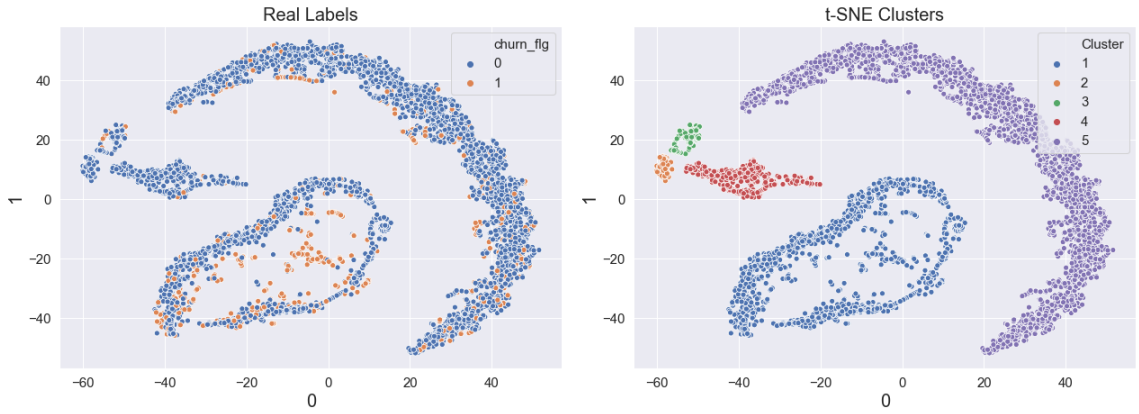
- Cluster 4 (Red): Contains customers that have Internet by optic fiber and their monthly values of download and upload are low, in the hundreds of KB, having some months where the value is 0 KB. Also, the values of the invoices are low when compared with the Clusters 1 and 5.

- Cluster 5 (Purple): Contains customers that have Internet by optic fiber and their monthly values of download and upload are high, achieving maximum values in the order of millions of KB, points at the bottom of the cluster, and minimum values in the order of tens of thousands at the top of the cluster.

4.2.3 Results

The software described in the previous chapter was again used to identify anomalies in this dataset with the parameter of tuning enabled and disabled for comparison. The level of contamination used was set as the percentage of client churners in the training dataset, which, due to the random train/test split was similar to the percentage of churners in the test set.

As previously mentioned, this dataset contains instances where the features download and upload were missing. This happens because some users do not have Internet by optic fiber, and the attributes of download and upload we had in our dataset only measure



a Involuntary Churn t-SNE with real labels

b Involuntary Churn t-SNE with the different clusters

Figure 4.11: Involuntary churn t-SNE with the real labels and the different clusters observed

the consumption for that type of Internet service. To deal with these missing values, we selected the option of inputting by a predefined value, and we chose that value to be -1, since this value is not present for download and upload features, that have values either zero or higher. All the values from the training and testing dataset were also scaled using a standard scaler, as explained in the previous dataset. When tuning is selected, the value of divisions in the K-cross validation was again set to two.

4.2.3.1 Tuning Gain

After the results with and without tuning were obtained, we analyzed the advantage of parameter tuning in the anomaly detection algorithms by plotting the resultant f1-Score of both cases. In figure 4.12 we see that the gains in f1-Score are significant and this holds for almost all algorithms. This was not present in the case of fraud detection, where the dataset had fewer features, and the problem was less complex.

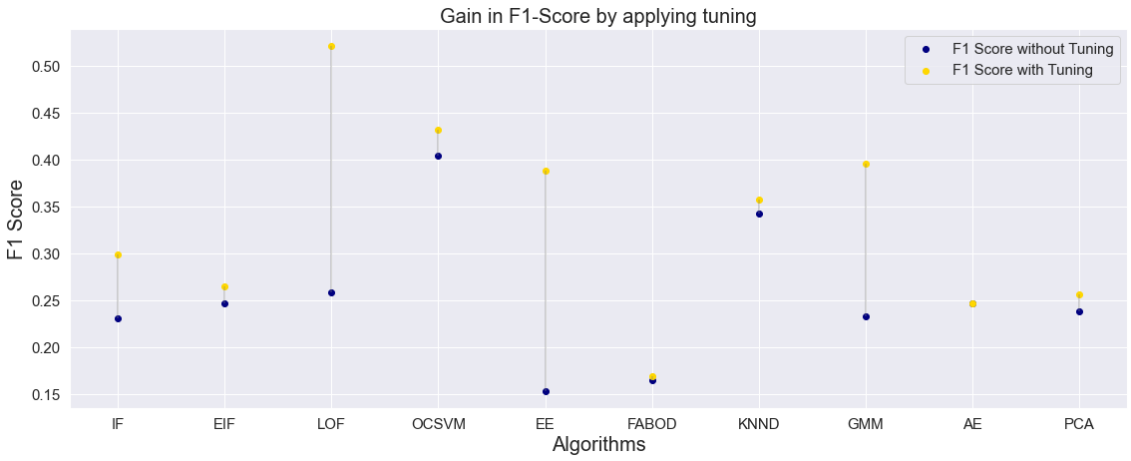


Figure 4.12: Involuntary churn tuning gain

Although in this case, only the Auto-Encoder did not have any f1-score gain, we will only discuss in detail the tuning of three algorithms (Local Outlier Factor, Elliptic Envelope, Gaussian Mixture Models) where the gain was significant.

Local Outlier Factor

The Local Outlier Factor algorithm has the highest gain in f1-score, going from around 0.25 in the non-tuned version to around 0.5 in the tuned one. This gain is due to the change of the number of neighbors from 20 to 92, and the change of the distance metric, as shown in table 4.7.

Table 4.7: LOF default and tuned parameters in involuntary churn dataset

Parameters	Contamination	Number of Neighbors	Metrics	Algorithm
Default	0.13	20	Euclidean	Auto
Tuned	0.13	92	Chebishev	Auto

Figure 4.13 showcases the tuning gain, with two t-SNE plots, the first one showing the labels predicted by the Local Outlier Factor algorithm with default parameters, and the second plot showing the labels predicted by the tuned model. These predicted labels can be compared with the real labels in figure 4.11a.

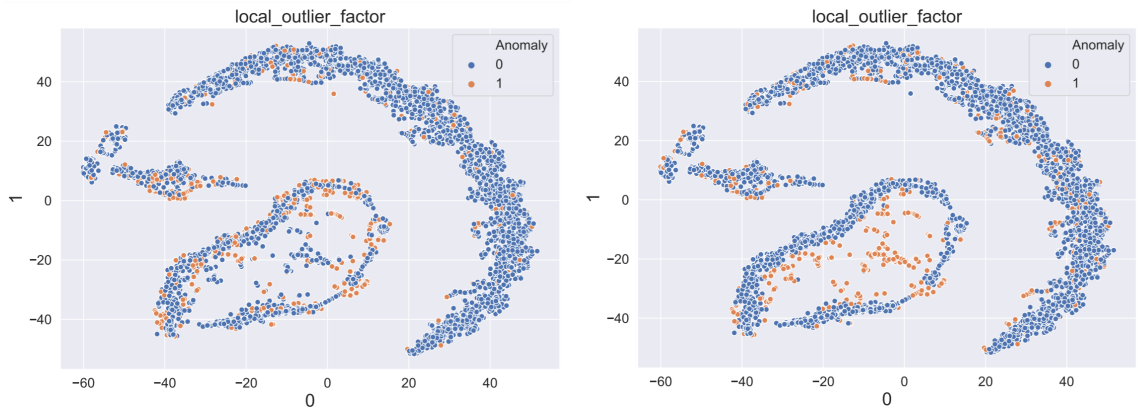
The major differences in the results are due to the points in the middle of the circular cluster that all seem to be identified as anomalous in the tuned model. Out of 399 real anomaly points in that cluster, referred to as cluster 1 in figure 4.11b, the non-tuned model found 127 anomalies while the tuned model found 231; with the improvement happening due to the rise in the number of neighbors.

Since the LOF algorithm assigns a higher anomaly score to points with lower density than the other points of its neighborhood, in the first model when the neighborhood size was set to 20, the model did not identify the scattered points inside of the circular cluster as anomalous. These points were being compared only to their immediate neighbors, other scattered points, not being identified as points with low density. By raising the number of neighbors to 92 these scattered points were now being compared, not only to the remaining scattered points, but also to the other points with higher density, resulting in their identification as anomalies. The data points being classified as an anomaly in the tuned model are mainly the ones corresponding to customers without Internet by optic fiber and with some invoices of the past five months with the amount set to 0.

Elliptic Envelope

The Elliptic Envelope algorithm also has a high gain in f1-score, going from around 0.15 in the non-tuned version to around 0.4 in the tuned one. This gain is due to the change in the support fraction used to train the model, from the fraction calculated using the default formula to the use of all the data points, as explained in section 3.2.5.

In figure 4.14 we demonstrate the gain in f1-score by showing the prediction results of both the tuned and non-tuned versions on t-SNE plots. As we can observe in the



a t-SNE with predictions without tuning

b t-SNE with predictions with tuning

Figure 4.13: Comparison of Local Outlier Factor predictions between the default and tuned versions in the involuntary churn dataset

Table 4.8: Elliptic Envelope default and tuned parameters in involuntary churn dataset

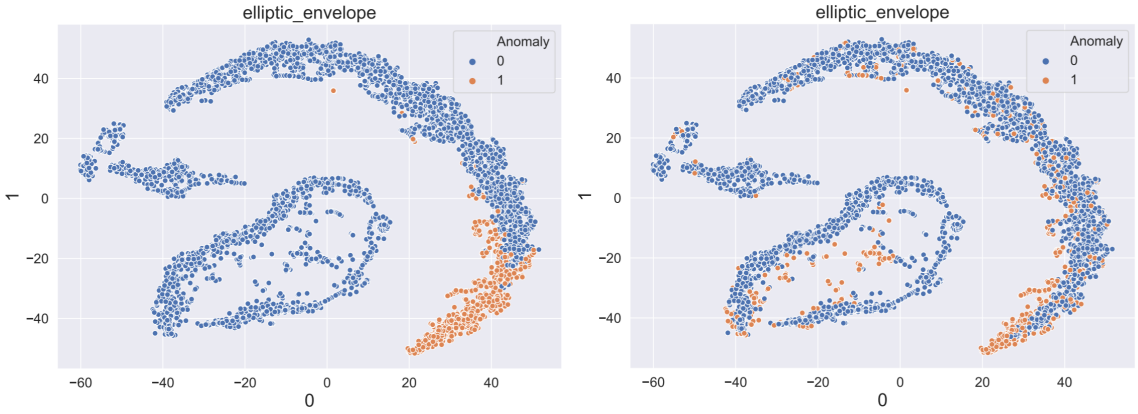
Parameters	Contamination	Assume Centered	Support Fraction
Default	0.13	False	None
Tuned	0.13	False	1

non-tuned version the model is detecting only the values in the bottom of the cluster 5, which as mentioned before, contains customers with high usage of Internet. Hence, the algorithm is detecting as an anomaly the extremely high values of usage. In the tuned version although still considering as anomaly the users with higher usage of Internet, the model was able to find other anomalous data points in other clusters. This could be because in the non-tuned model, the chosen support fraction was set to 0.5 of the training dataset, which was not sufficient to capture enough information about the distribution of normal data. After tuning, the whole training dataset was used which led to the improvement of the performance.

Gaussian Mixture Models

In the tuning process, three of the parameters were changed to obtain a better f1-score, just like for the previous dataset. One of the parameters was the number of Gaussian components which changed from 5 to 4, meaning that this dataset has a lower number of different distributions than the default value. Also, the covariance type and the parameter initialization method were changed. The table 4.9 contains the parameters used in the default and tuned cases.

As explained in the previous section of the credit card fraud dataset, the gains in f1-score when using the Gaussian Mixture Models algorithm are usually related to the choice of the number of Gaussian components. Without tuning, the number of Gaussian distributions fitted to the data was higher, causing the model to consider the anomalous



a t-SNE with predictions without tuning

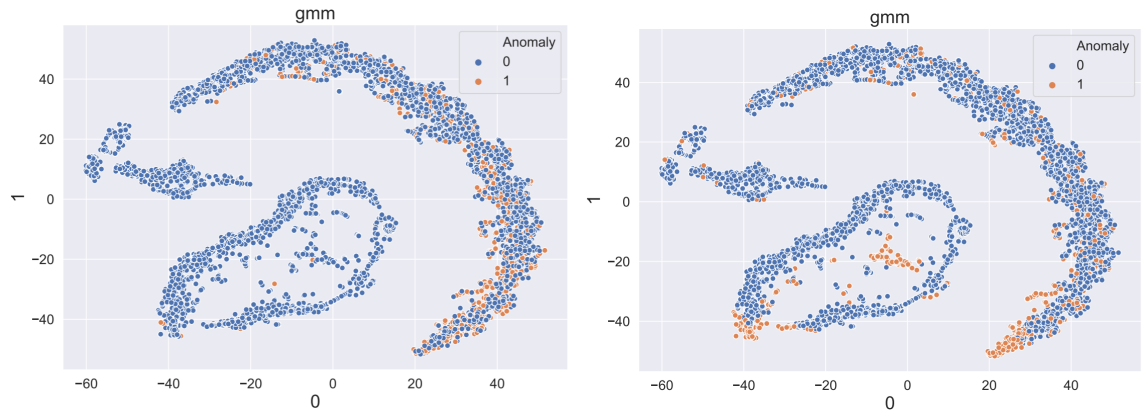
b t-SNE with predictions with tuning

Figure 4.14: Comparison of Elliptic Envelope predictions between the default and tuned versions

Table 4.9: Gaussian Mixture Models default and tuned parameters in involuntary churn dataset

Parameters	Contamination	Nr. of Components	Covariance	Initialization
Default	0.13	5	Full	K-means
Tuned	0.13	4	Spherical	Random

points inside cluster 1 as a part of a component, instead of considering them as anomalies, which is what happened in the tuned version. We can observe this in figure 4.15.



a t-SNE with predictions without tuning

b t-SNE with predictions with tuning

Figure 4.15: Comparison of Gaussian Mixture Models predictions between the default and tuned versions in involuntary churn dataset

4.2.3.2 Performance Metrics

In the table 4.10 we summarize the performance of different anomaly detection algorithms applied to the dataset of involuntary churn. Further below, we will discuss different metrics of performance.

As we mentioned in the table 4.6, 28% of the instances have missing values in four of the attributes, and we have imputed these missing values with the constant value -1. Hence, we could not analyze the performance of the Fast ABOD algorithm, since this algorithm cannot be used for datasets where features have low variance. Given that the 28% of data points ended up with the same value for 4 features, there were points whose neighbors had the same values of features, and distances were too small for this implementation of Fast ABOD to function.

Time

The differences in the magnitude of the values for the time taken by each algorithm are proportional to the values of the previous dataset. This happens because the two datasets have a similar number of instances, and the fact that the second dataset has slightly more attributes does not seem to have a significant impact on the computation time.

Table 4.10: Performance of different anomaly detection algorithms for the involuntary churn dataset

Algorithm	Time	Lift_1	Lift_10	Roc_Auc	PR_Auc	F1	Recall	Precision	Accuracy
isolation_forest	94	1.85	2.39	0.699	0.246	0.299	0.294	0.304	0.817
ext_isolation_forest	309	1.19	1.92	0.719	0.236	0.265	0.261	0.270	0.808
local_outlier_factor	61	5.30	4.70	0.800	0.473	0.521	0.513	0.530	0.875
one_class_SVM	323	2.52	3.25	0.758	0.328	0.432	0.425	0.439	0.852
elliptic_envelope	100	1.85	3.04	0.770	0.322	0.388	0.382	0.394	0.840
knnd	202	2.52	2.80	0.758	0.303	0.357	0.352	0.363	0.832
gmm	23	1.46	3.46	0.755	0.318	0.396	0.390	0.402	0.842
auto_encoder	2390	1.46	1.76	0.679	0.215	0.247	0.243	0.251	0.803
pca	1	1.46	1.73	0.677	0.214	0.256	0.252	0.260	0.806

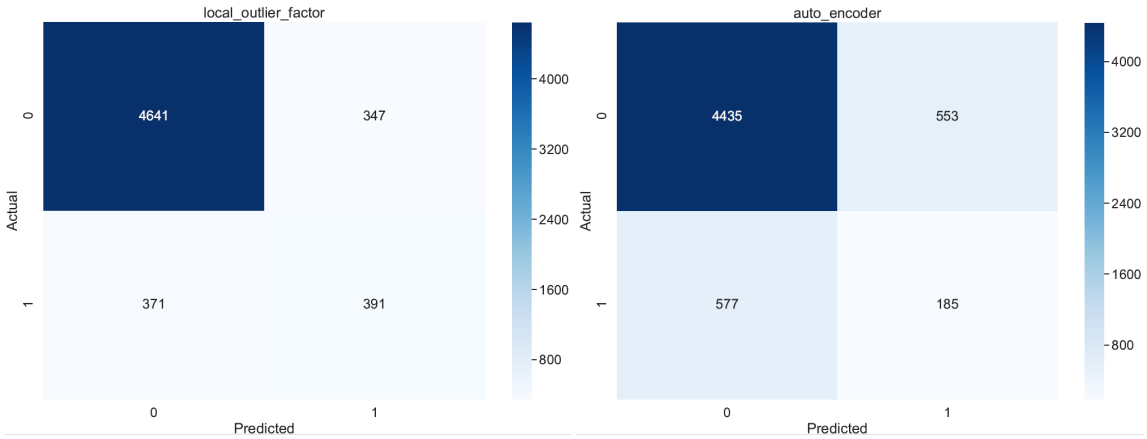
F1-Score, Recall and Precision

The algorithm with the best f1-score is the Local Outlier Factor with a score of 0.521, and this is also the algorithm with the best Recall and Precision, meaning that it identified the highest number of churners.

As seen in the t-SNE plot in figure 4.11a the anomaly points are sometimes very close to normal points, explaining the low value for the Recall and Precision even of the highest performing algorithm.

Figure 4.16 shows the confusion matrix of the algorithm with the highest score, Local Outlier Factor, and the one with the lowest score, Auto-Encoder. We can observe that the LOF model identified a higher number of churners, but also, it made fewer errors of labeling churners as non-churners, and vice versa. We additionally illustrate this difference in figure 4.17 with two t-SNE plots, side by side, of the predictions of each of these algorithms. The difference in f1-scores can be explained by both algorithms considering as anomaly, points with different characteristics. T-SNE plots with the results of the other algorithms are given in appendix A.

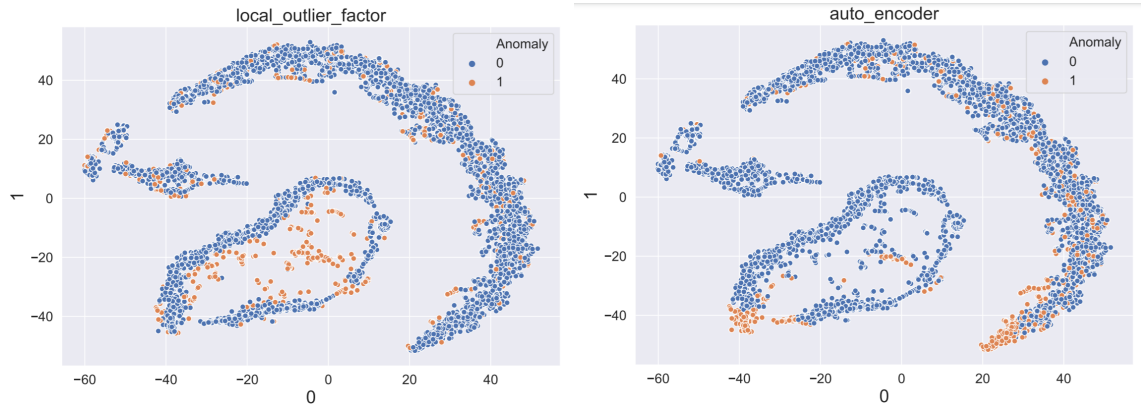
As explained before in the tuning gain section 4.2.3.1, LOF algorithm is detecting data points that are spread out and have less density than their neighbors, which is the reason why it is correctly detecting the data points inside the circular cluster and not the ones in the extremities of the clusters, as they have high density. The Auto-Encoder is detecting data points of customers in the extremities of the two main clusters, cluster 1 and 5. In the case of the circular cluster or cluster 1, which represents customers without Internet by fiber optic or with low usage of Internet, it is detecting the data points in the bottom which correspond to customers with high invoice amounts. In the case of the semicircular cluster or cluster 5, which represents customers with high values of Internet consumption, it is detecting the data points with the higher values of downloads and uploads.



a Local Outlier Factor Confusion Matrix

b Auto-Encoder Confusion Matrix

Figure 4.16: Comparison between the confusion matrix of Local Outlier Factor and Auto-Encoder in the involuntary churn dataset



a t-SNE with predictions of Local Outlier Factor

b t-SNE with predictions of Auto-Encoder

Figure 4.17: Comparison between the t-SNE plots of Local Outlier Factor and Auto-Encoder for the involuntary churn dataset

ROC-AUC and PR-AUC

Looking at the values in the table 4.10, we observe that there is greater variability between the performance of different algorithms. A higher number of features, as well as higher difficulty of this problem, could explain this variability which was not present for the previous dataset. Figure 4.18 shows the ROC curve plots of all the algorithms. We can observe why we could not consider the results for the Fast ABOD for this dataset: its ROC curve almost resembles a straight line. We can also observe in the figure 4.18 that the algorithm with the best results of ROC-AUC is the Local Outlier Factor as it had been for the f1-score, recall and precision. This is also true for the PR-AUC as we can see in table 4.10. In terms of ROC-AUC and PR-AUC, the worst algorithm is the Principal Component Analysis and not the Auto-Encoder as it happened in the f1-score, recall, and precision metrics. Still, the Auto-Encoder presents the second-worst results, so we can conclude that in this dataset the values of the area under the curves are agreeing with the values of the other metrics.

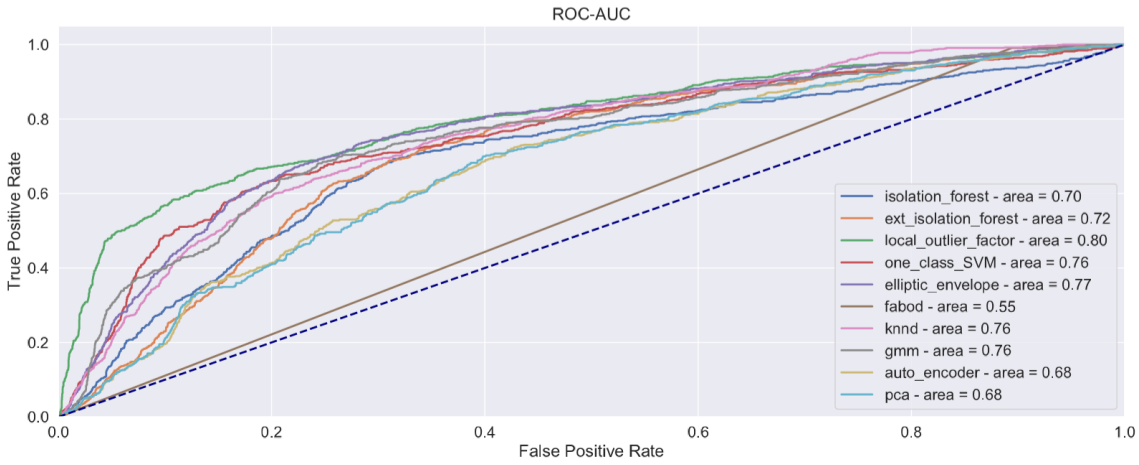
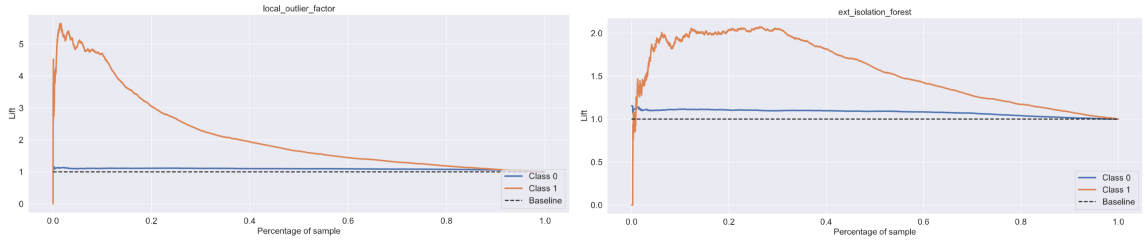


Figure 4.18: Involuntary churn ROC Curves

Lifts

Next, we discuss the lift for the first percentile and the first decile, again looking at the values in the table 4.10. Same as with the other metrics, the lift gains in this dataset are much smaller than the ones in the credit card dataset, as already discussed. Still, all the lift values are above 1, meaning that choosing values in the first percentile and the first decile of the ranked anomaly scores has advantages, in comparison with choosing randomly. The algorithm with the highest lift gain in both the first percentile and the first decile is the Local outlier Factor with values of 5.30 and 4.70 respectively. This means that if we randomly selected 1% of the test dataset we will have 58 data points, out of which 8 are anomalous. However, if we look at the 1% data points with the highest anomaly scores, we end up having around 41 anomalous data points out of the 58 data points, which is 5 times higher than random. The algorithm with the worst lift gain in the first percentile is the Extended Isolation forest with a lift of 1.19. The lift for the first decile is higher than the one for the first percentile and this can be explained by some false positives in the data points with the highest anomaly score. In figure 4.19 we can see the difference between the best and worst lift. Both plots start with gains lower than one since both algorithms attribute the highest anomaly scores actually to normal points. By comparing the t-SNE plots of Local Outlier Factor and the Extended Isolation Forest predictions in the first percentile in figure 4.20 we can understand why the lift gains of both algorithms are different. While the Local Outlier Factor is detecting the scattered anomalous points in the first percentile, the Extended Isolation Forest is detecting the points in the extremities of the clusters. As we concluded before, the majority of anomaly points are scattered in the center of the circular cluster.



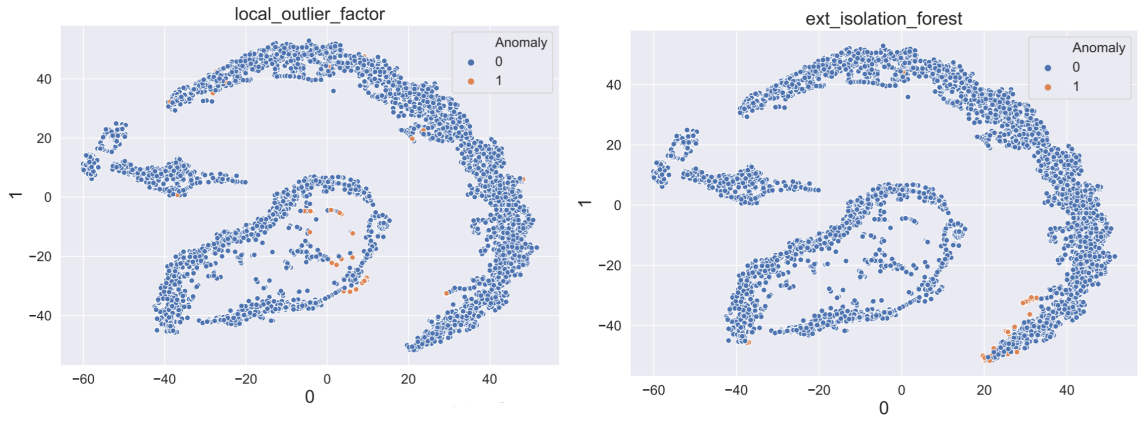
a Local Outlier Factor Lift

b Extended Isolation Forest Lift

Figure 4.19: Comparison of lift plots between the Local Outlier Factor and Extended Isolation Forest predictions

Excess-Mass and Mass Volume

We did not include the results for the Excess-Mass and Mass-Volume curves for this dataset, since these metrics in their basic form can only be used in low-dimensional datasets, usually under seven attributes. Hence, the results of using these algorithms were not reliable.



a Local Outlier Factor t-SNE plot

b Extended Isolation Forest t-SNE plot

Figure 4.20: Comparison of t-SNE plots between the Local Outlier Factor and Extended Isolation Forest predictions in the first percentile

4.3 Conclusions

By analyzing the performance of different algorithms on these two datasets we can conclude that there is not a single algorithm that is the best for all the cases, neither for all the metrics nor for all the datasets. Therefore, the correct approach for selecting the best algorithm would be to test several algorithms and choose the most appropriate one for our problem. The software that we developed could help, not only by aiding in the choice of the algorithm, but also in the choice of the best metric to evaluate the performance of the algorithms. For example, lifts might be a good choice for addressing the churn problem because the operators usually have a limit on how many people they can contact. If the operator wants to maximize that the number of potential churners found in the percentage of clients that can be contacted, then the model with the highest lift for that percentage should be used. However, in the case of credit card fraud, each fraud is very costly to the client and the bank, while the false positives might not be as costly, so we may choose the algorithm with the best recall. If a false positive also implies major inconveniences to the customers, then f1-score might be a more adequate metric. Additionally, the software could be very useful at the beginning of a project, where it can be used on a subsample of the dataset, to get the idea of what could be the most promising approaches and parameters for a specific problem.

Chapter 5

Detecting Low Quality of Service

In this chapter, we address an even more challenging problem of identifying periods when customers experience low quality of Internet service. One of the reasons which contribute to making this a challenging problem is that no accurate information exists on when exactly did customers experience low quality of service (QoS). What is available are details on customers' complaints about the service, and these complaints mostly coincide with periods of low QoS. However, there exist multiple issues regarding usage of these complaints as labels for identifying anomalous events. First, there is no guarantee that customers report immediately the issue as soon as they experience low QoS. Instead, there might be a lag, different for each customer. Also, the efforts by the telecom operator to restore service quality after the complaint might cause additional anomalous events. Furthermore, not all customers call to complain even though they experience such low QoS, or some might call regarding an issue that is not reflected in the measurements present in the dataset that we will use to address this problem. Each customer might have their own level of tolerance for service quality, and each customer might have different equipment, hence there is no objective measure of what is an acceptable service quality for each customer. Another reason why this problem is different from the other two that we have discussed in the previous chapter is that the measurements of service quality are taken over time. Hence, our data is sequential, and different methods can be used for anomaly detection to explore this specific data structure. Additionally, as we will detail below, this dataset has many missing values, which are not simply missing at random, but in some cases, they are also a consequence of low QoS.

We apply two different approaches to this problem of identifying low QoS. In the first approach, we apply the developed software, without exploring the time series component of the dataset. In the second approach, we apply and optimize an algorithm tailored for anomaly detection in time-series, LSTM Auto-encoder described in section 2.3.3.2.

Therefore, in the remainder of this chapter, first, we will explain the dataset and its features. Then we will discuss the performance of the algorithms incorporated in the developed software applied to this problem. Next, we will describe the approach based on LSTM Auto-encoder and present the results obtained with these approaches: the basic setup, the choice of parameters, as well as the inclusion of additional non-sequential information, and imputation of missing values.

5.1 Quality of Service Dataset

The quality of service dataset that we will be using contains a history of quality of service-related features from 10,000 customers of a telecommunications company from 03/09/2019 to 30/09/2019. These features are as follows:

- downstream power on the receiver,
- upstream power on the receiver,
- upstream power on the transmitter,
- downstream and upstream Codeword Error Rate (CER),
- downstream and upstream Corrected Codeword Error Rate (CCER),
- downstream and upstream signal-to-noise ratio and
- quantity of bytes downloaded and uploaded.

In addition to the quality of service variables, each data point contains the information of the MAC address of the customer to which it belongs, the day and time that the variables were recorded, and also variables corresponding to the model of the customers' device and of the cable modem termination system (CMTS) of the customer. These additional variables were not used in the first stage, when we applied the classical anomaly detection algorithms from the software, since they did not have a significant impact on the results in the preliminary analysis. However, these variables were later used as additional information for LSTM Auto-encoder, which we will discuss later on.

The granularity of this dataset is hourly, where, in the case of the downloaded and upload bytes this corresponds to the total number of bytes transmitted in that hour and for the other variables, it corresponds to an hourly average. Therefore, each data point in this dataset is an hourly measurement of 11 metrics of service quality for a specific customer. For each hour we have 10,000 data points, each one corresponding to a different customer. For each customer, we have hourly measurements for the period of approximately 4 weeks. Each day of data has only 21 hours since the dataset does not include the measurements between 4 and 6 p.m., due to maintenance activities of the monitoring system during these hours. As an auxiliary feature, this dataset also contains an hourly signal score

which rates the quality of signals in this hour, using a predefined set of rules. The best score is attributed to instances when no signal steps out of the thresholds defined by the technicians and the worst score is attributed to instances where all signals are outside the thresholds. Instead of a label, we have additional information whether the customer made a complaint related to malfunction during that hour. Since there is uncertainty associated with the labels, as explained above, to evaluate the efficiency of the algorithms we will consider as a true positive any data point predicted as an anomaly if it is in the range from five hours before the complaint and up to five hours after the complaint. As a false positive we will consider any data point predicted as an anomaly that is outside these eleven hours. These values were chosen based on exhaustive preliminary data analysis. Another aspect that makes this dataset very challenging, is the existence of missing values in the data points, with 11% of the data points having at least one of the attributes with a missing value. These missing values can appear in all the features of the same data point at once, meaning that probably there was a problem while retrieving this data from the customers. Also, there are missing values in isolated features or subsets of features.

5.2 Classical Anomaly Detection in the Context of QoS

In this section, we present the performance of the algorithms for point anomaly detection that are included in the developed software. Again, the Fast Angle-Based Outlier Detection algorithm was not used because it is very time costly and not much effective, as shown in the previous chapter. Since we had 11 measurements from 21 hours per day for 10,000 customers during a period of 28 days, giving in total 5,880,000 data points, we sampled the dataset to decrease the running time of the software.

5.2.1 Data Pre-Processing

Sub-Sampling: To reduce the size of the dataset, we chose for the training dataset the data points from 03/09/2019 to 23/09/2019 from 50 customers. Twenty-five of those customers were chosen because they had the best signal scores and some algorithms need normal data points to train their model. The remainder 25 were customers with complaints during that period, and they were included to provide examples of anomalies in the training dataset for parameter tuning. For the testing dataset, the data points from 24/09/2019 to 30/09/2019 were selected and all the 30 customers with complaints in that time frame were included. We selected these periods for training and test set specifically to have no overlap between them, even though no customer was present in both the training and testing set. The reason is that we wanted to make sure there was no data leakage, that is when the data we are using to train a machine learning algorithm happens to have the information we are trying to predict, which could happen in case of a global service failure, which could impact more than one customer at the same time. In this case, if we used some customers impacted by a global anomaly in the training

set and others in the test set, we could obtain a possibly overly optimistic estimate of the algorithms' performance. As we explained previously, in the case of the complaint, we labeled 5 hours before and after the complaint as anomalous, leaving still plenty of normal data points in our test set.

We summarize the most important aspects of the resulting dataset in table 4.6, indicating the total number of data points and features, the number of samples and features containing missing values (NA), the level of anomaly contamination in the dataset, and the train/test split percentages.

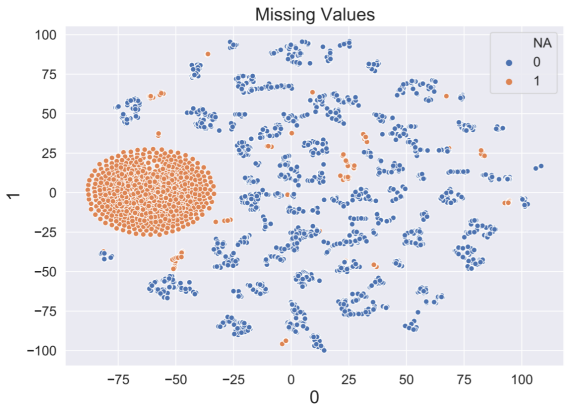
Table 5.1: Quality of Service Dataset Overview

Number of Samples	Samples with NA	Number of Features	Features with NA	Contamination [%]	Train/Test Split [%]
26460	3078	11	11	2	83/17

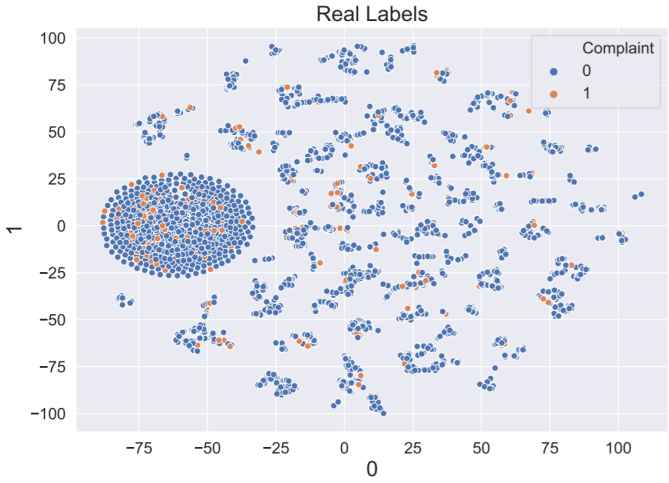
5.2.2 Data Visualization

To understand better how the anomalies of this dataset are distributed and to gain other insights about the distribution of the data points, we once again generated a t-SNE plot with the value of perplexity set to 10, which was the value that gave the best separation between the data points from the multiple customers in the testing dataset. In figure 5.1a we have a t-SNE plot where each one of the colors corresponds to one of the 30 customers in the training dataset. The formed clusters are predominantly of the same color, meaning that data points from the same customer are similar to each other, as expected. Also, as expected, there is a bigger cluster with data points from several different customers that correspond to the data points where all the feature values are missing, i.e., no measurement is available. We can confirm this by looking at the figure 5.1b, where we used the same t-SNE plot, but now we colored orange all the data points that have at least one missing value. As said before, the data points in the bigger cluster represent the data points where all the feature values are missing, and the scattered orange data points are the ones where only some attribute values are missing. We also generated a t-SNE plot with the labels of complaint, as shown in figure 5.1c. Here, we marked as anomaly 11 points per customer, which correspond to data points up to 5 hours before the complaint, the hour of the complaint, and up to 5 hours after the complaint, as described previously. In this figure, we can observe that the anomaly data points are scattered in the plot, because there are many different reasons for a customer to make a complaint, and some of these reasons might not be reflected in these features. Additionally, we are using 11 data points as anomalies for each complaint, and some of these data points could actually be normal points. There are a great number of anomaly data points in the big cluster composed of data points with missing values, meaning that probably the missing values in some cases are a consequence of a malfunction that generated a complaint.

a t-SNE plot of the QoS dataset with data points colored according to which customer they belong to



b t-SNE plot of the QoS dataset with data points colored according to whether they have missing values



c t-SNE plot of the QoS dataset with complaint labels

Figure 5.1: t-SNE plots of the QoS dataset

5.2.3 Algorithm Performance

The data points of the training and testing dataset that had missing values were imputed using the software option of imputation by extreme value. This option performs *minmax* scaling, that scales all data points to a range of 0 to 1, and then replaces all missing values with −0.1., as explained in 3.1.1. Afterwards, standard scaling of the training and testing dataset was performed. The training dataset was used for parameter tuning and training of the algorithms, and then the models were applied to each one of the 30 test customers with a complaint. Once the anomaly scores were obtained as the output of the algorithm, we did not use the contamination level of the training set to convert the scores to labels. Instead, we labeled as anomaly those data points with anomaly score higher than the mean plus three times the standard deviation of the customer anomaly score. We had this threshold set per customer, to compare the results with those from the LSTM based approach, though the threshold could be also defined globally. The reason we resorted to this approach of setting the anomaly threshold is that it is often used in

anomaly detection problems, and in our preliminary analysis, it led to better performance than the threshold based on the contamination level. Since the dataset did not include the information on the exact starting time of the problem that originated the complaint, we consider as a complaint 5 points before the complaint hour, the data point in the complaint hour, and the 5 after the complaint hour, as explained before. However, to evaluate the algorithms using different performance metrics without having too many false negatives, we adjusted the labeling in such a way that if the algorithm classified as anomaly at least one of the 11 data points with anomaly label, we counted all those 11 points as one true positive. If the algorithm did not classify as anomaly any of the 11 data points, we counted this as one false negative. All the data points classified as an anomaly that were actual normal points were considered as false positives.

We removed the results from the Gaussian Mixture Models algorithm because it was not able to generate a good decision function; it classified the majority of the data points with the same anomaly score.

Next, we discuss in detail the results in table 5.2, obtained for the above setup.

Time

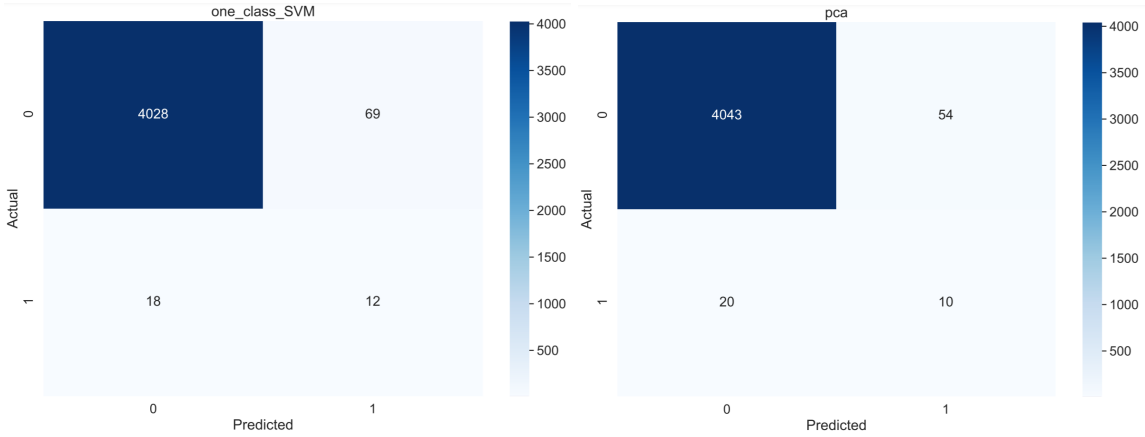
Since the training dataset used in this chapter was nearly 4 times bigger than the previous two datasets, the time taken for running the software was generally larger. Only the PCA took the same time and that is because the parameter tuning for this algorithm has a small search space, so the rise in the data points number did not influence the execution time that much. Although most of the algorithms had a proportional increase of their running time compared to the previous cases, One-Class SVM had a major rise in the running time, nearly 20 times bigger than for the previous datasets. This is because SVM training scales badly with the number of samples, typically $O(n^2)$.

Table 5.2: Performance of different anomaly detection algorithms for the quality of service dataset

Algorithm	Time	Mean Lift_1	Mean Lift_10	Mean Roc_Auc	Mean PR_Auc	F1	Recall	Precision	Accuracy
isolation_forest	198	9.13	2.82	0.647	0.143	0.188	0.267	0.145	0.983
ext_isolation_forest	836	22.83	4.57	0.677	0.237	0.178	0.400	0.114	0.973
local_outlier_factor	162	13.70	3.50	0.640	0.169	0.163	0.267	0.118	0.980
one_class_SVM	6032	9.13	4.57	0.742	0.153	0.216	0.400	0.148	0.979
elliptic_envelope	45	31.96	4.22	0.688	0.283	0.203	0.400	0.136	0.977
knnd	258	13.70	3.87	0.708	0.177	0.164	0.367	0.106	0.973
auto_encoder	2732	4.57	3.87	0.720	0.161	0.202	0.333	0.145	0.981
pca	1	18.33	3.87	0.690	0.312	0.213	0.333	0.156	0.982

F1-Score, Recall and Precision

In terms of f1-score, all algorithms achieved similar results, with One-class SVM having the highest score of 0.22. These values of f1-score are really low when compared with the values from the previous datasets, the reason being the difficulty of the dataset. As explained before there are many different reasons for a customer to make a complaint and the labels used to evaluate the models are inexact. Also, this dataset has a time-series sequential component that is not being explored by the algorithms present in this software. Although One-Class SVM had the best f1-score, as well as recall, PCA algorithm had the highest precision, meaning that it had the highest percentage of actual anomalies in the points it identified as anomalous. We can observe this by looking at the two confusion matrices in figure 5.2, where One-Class SVM identified 12 anomalies out of 30, whereas PCA identified only 10, however PCA had less false positives, which resulted in its higher value of precision.

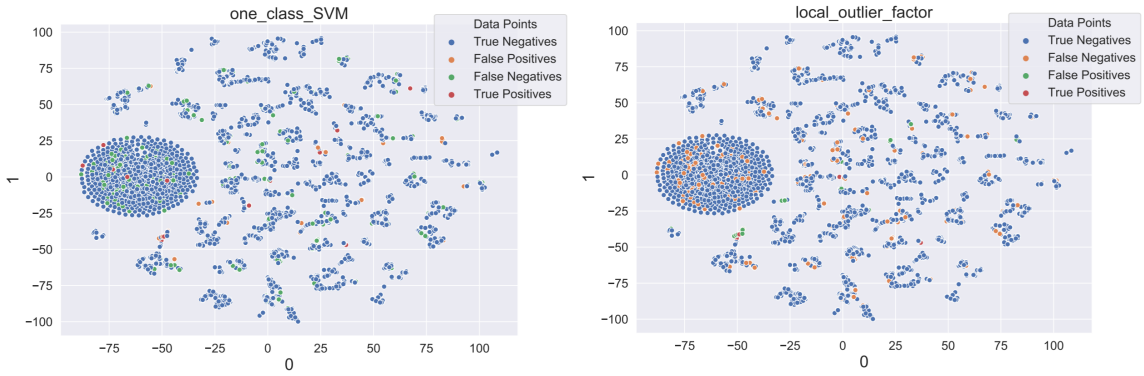


a Confusion Matrix of OCSVM

b Confusion Matrix of PCA

Figure 5.2: Comparison of the confusion matrices of One-Class SVM and PCA for the QOS dataset

We also compared the data points considered an anomaly by the One-Class SVM, the best algorithm, and the ones considered as an anomaly by the Local Outlier factor, which is one of the two worst algorithms. In the t-SNE plots in figure 5.3, we presented which points did each one of the algorithms correctly classify and which ones it misclassified, still considering 11 anomalous data points by customer. We can observe that Local Outlier Factor has less true positives than the One-Class SVM, because, as explained in chapter 4, Local Outlier Factor finds anomalies in scattered data points with lower densities than its neighbors. In this case, a great number of anomalies are located in the bigger cluster containing points with missing values, and in this cluster all the data points are close together. Hence, Local Outlier Factor does not identify those points as anomalies. This is not the case for the One Class-SVM, which identified 5 real anomalous data points from that cluster.



a t-SNE of One-Class SVM results

b t-SNE of the Local Outlier Factor results

Figure 5.3: t-SNE plots of the results of One-Class SVM and PCA for the QoS dataset

As mentioned in the previous section, we set the threshold separately for each customer, and then we evaluated the performance individually for each customer. The results shown in the table 5.2 for the lifts, ROC-AUC and PR-AUC represent an average for all the customers, with some customers having higher, and some lower values of these metrics. Since these results were not satisfactory, we proceed with the approach more adequate for sequential data, an approach which would allow us to take advantage of the special structure of this dataset.

5.3 LSTM Based Anomaly Detection in the Context of QoS

Next, we describe the LSTM Auto-Encoder we used to predict anomalies in the Quality of Service data. As discussed in subsection 2.3.3, LSTM Auto-Encoder represents state of the art for detecting anomalies in time-series data. First, we explain the basic architecture of an LSTM Auto-Encoder, then we vary its parameters to study their impact on the performance, and choose the optimal ones by building an individual model for a customer. Afterwards, we train a single LSTM Auto-Encoder using measurements of multiple customers with good signal scores and without complaints and apply the model on the customers that made a complaint in the last week of data that we have. Lastly, we compare two approaches for dealing with missing data and study the value of additional contextual information: hardware type of the router and hub to which it is connected. We add this non-sequential data to time-series to see whether it has a positive impact on the model performance. The intuition is that the decision whether a measurement is normal might depend on these two attributes.

5.3.1 LSTM Basic Architecture

As we already discussed, LSTM auto-encoder is a type of autoencoder built with LSTM neural networks. This type of network accepts a specific type of input shape that is an array with 3 dimensions: batch, timestep and input dimension, as illustrated in figure

5.4. The value of timestep corresponds to the dependency of a data point on its previous data points, i.e., it defines how many previous measurements should be used to predict the value of the next data point. Batch size represents the number of different groups of timesteps that will be fed to the LSTM to make an update of network weights, and the input dimensions are the number of attributes of the dataset.

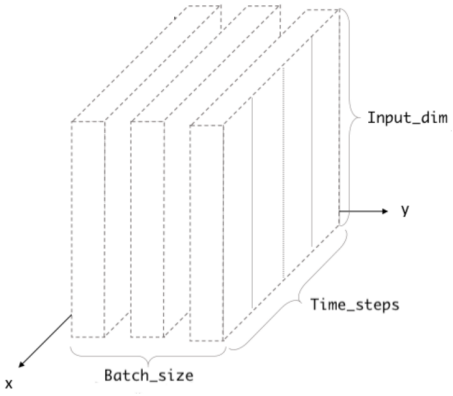


Figure 5.4: Input shape of an LSTM neural network [22]

We varied the number of timesteps, as well as the batch size to see which values lead to better performance. To transform the original dataset to the specific format required by the LSTM network, data from different customers was placed in groups with the size of the variable *timestep*, as shown in figure 5.5. There, for illustration purposes, we have a multivariate time series dataset with two attributes represented with the colors blue and yellow. Input sequence is divided into n overlapping sliding windows, whose size corresponds to the value of *timestep*.

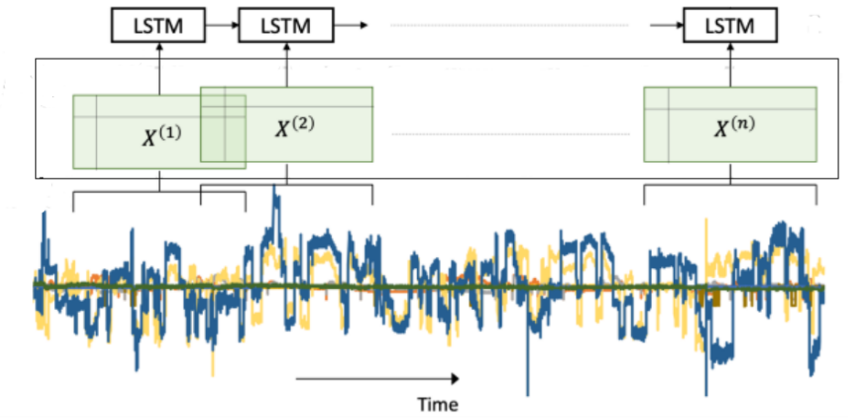


Figure 5.5: Transforming the dataset into LSTM input shape [35]

To select the best parameters for the task of identifying periods of low quality of service, we used the data points from 03/09/2019 to 23/09/2019 to train the LSTM Auto-Encoder. The data points from 24/09/2019 to 30/09/2019 we used as test, and we replaced the missing values with a value outside the normal range. The reason for constructing the dataset this way is the same as when we applied point anomaly algorithms

from the developed software, and this is explained in section 5.2.1. Additionally, later on, we compare the performance of this type of neural network tailored for time-series data and the classical point anomaly algorithms.

5.3.1.1 Optimizing the Number of Timesteps for Input

For choosing the number of timesteps to be used in the LSTM Auto-Encoder architecture, we tested different values and compared the resulting LSTM performance. The value of timestep is an important parameter because it defines the number of previous data points that should be used to predict the next data point. We tested six different values: 7,14,21,28,35,42. We chose these values because one day of our dataset has 21 hours, so this way we can test values of dependency between the data points from one-quarter of a day up to 2 days. While tuning the timesteps, the activation function chosen was the 'ReLU', the number of neurons in the hidden layers were chosen as 8 in the first hidden layer and 4 in the second hidden layer, and the batch size was fixed at 32. In figure 5.6 we can observe the variation in accuracy, f1-score, precision, recall, time, and validation loss for different values of timestep. The values of time and validation loss were normalized between 0 and 1, so that we could plot all the metrics in the same figure. The timestep value of 7 has the best results in the 4 metrics (accuracy, f1-score, precision, and recall) , as well as the smallest amount of execution time and the lowest validation loss. Although the time is not important in this parameter tuning where only one customer is tested, when we add more customers to the LSTM Auto-Encoder, the time consumption of using a large number of timesteps will become prohibitive. Another insight of figure 5.6 is the correlation between the validation loss and the performance metrics, the metrics values are worse when using timesteps that generate a higher validation loss, as expected.

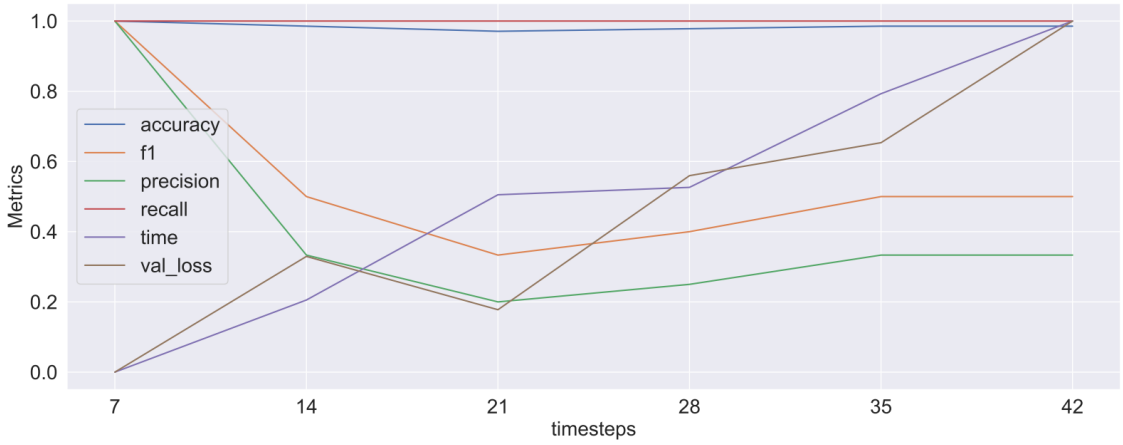
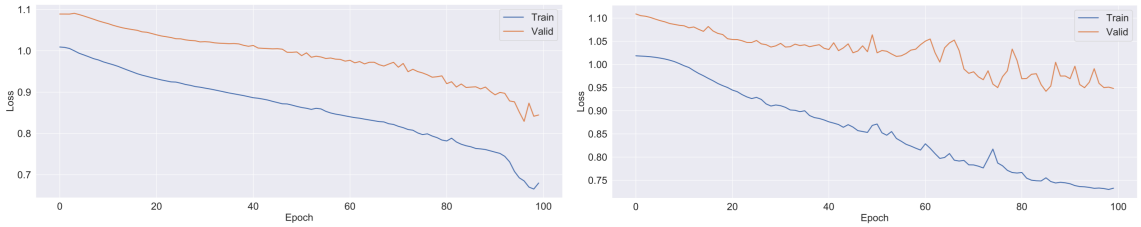


Figure 5.6: LSTM Auto-Encoder metrics with timestep variation

Another important aspect is the ability of the LSTM Auto-Encoder to correctly reconstruct the data. In figure 5.7 we can observe the difference in validation loss when using the value of timesteps of 7 and 21. In the first case, because the time dependency of each data point is smaller than in the second case, the neural network can better represent

the behavior, resulting in a smaller and less erratic validation loss. This can be caused by the majority of features in the dataset being signals of power and signal-to-noise ratio, and these signals depend much more on recent previous measurements, and not on measurements from 20 hours.



a Train and Validation Loss using 7 as the timestep b Train and Validation Loss using 21 as the timestep

Figure 5.7: Comparison of the LSTM Auto-Encoder losses using 7 and 21 timesteps

The LSTM Auto-Encoder anomaly score is defined by the mean squared error (MSE) of the Auto-Encoder prediction compared with actual values. In figure 5.8 we show the MSE for the duration of 7 days for 3 different values of timesteps, where the error for each value of timesteps is shown in different color. In the figure we also show the respective threshold for each value of timesteps, that is calculated as the mean plus three times the standard deviation of the error. The time of the complaint is represented as two vertical red lines on the same plot. From the figure we can see that at the time of the complaint the anomaly score when using 7 timesteps detects two anomalies of the 11 hours of complaint and does not predict any anomaly outside the time of complaint. This can be confirmed by the figure 5.6 where when using the value of 7 for timesteps, the precision is 1 because it did not have any false positives, and the recall is 1 because it found at least one anomaly in the 11 hours around the time of the complaint. The configuration with 14 timesteps and 21 timesteps also predicts an anomaly in the complaint hours, but at the cost of having 2 false positives in the first case and 4 false positives in the second case. Taking everything into account, using 7 timesteps is the ideal for this dataset and this makes sense intuitively since variables as signal-to-noise ratio and the number of downloads and uploads have a shorter time dependency, i.e., they are more influenced by the recent measurements, more than than the measurements that happened a long time ago.

5.3.1.2 Optimizing Batch Size

Another parameter that can influence the output of the LSTM Auto-Encoder is the batch size used in the training. This parameter can highly influence the time that takes to train the network, because the larger the batch size used in each iteration, the faster the training will be. However, lower batch size makes it easier to fit one batch worth of training data in memory and it may lead to lower generalization error. For tuning this parameter, we used a search space of 7 values: 8, 16, 32, 64, 128, 256, 435, were 435 represents the

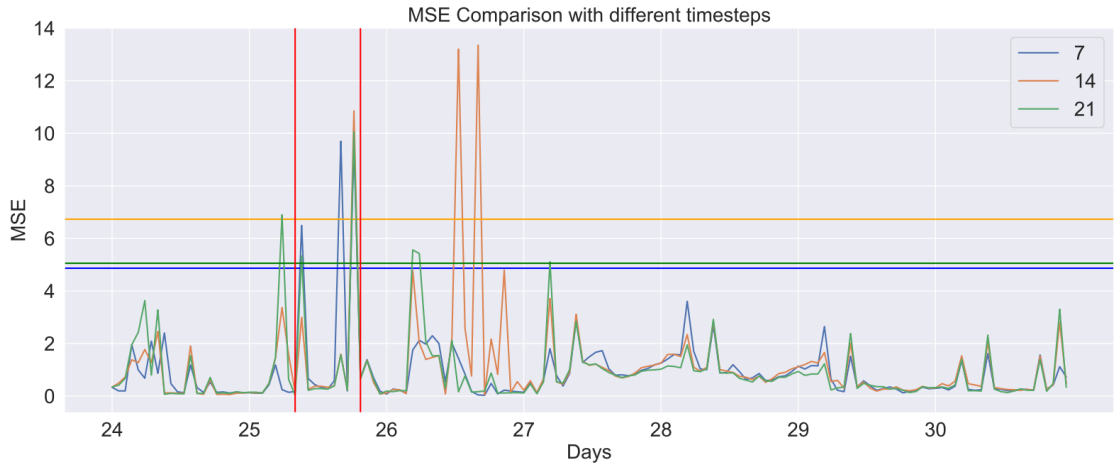


Figure 5.8: LSTM Auto-Encoder Anomaly Score with different Time Steps

total number of transformed data points in the training set. We also fixed the value of timesteps to 7 because it achieved the best results in the previous tuning. The number of neurons in the hidden layers was chosen as 8 in the first hidden layer and 4 in the second hidden layer and the chosen activation function was the 'ReLU'. The results of the tuning process are shown in figure 5.9, where with the increase in batch size, the time that it takes to train is reduced and the validation loss increases, as expected. To note that all the f1-scores in this tuning were equal to 1, meaning that all of the batch sizes used found at least one anomalous data point in the complaint hour and did not have any false positives. This could be explained by all the different versions using the same value of timestep, that is the most important parameter in the LSTM.

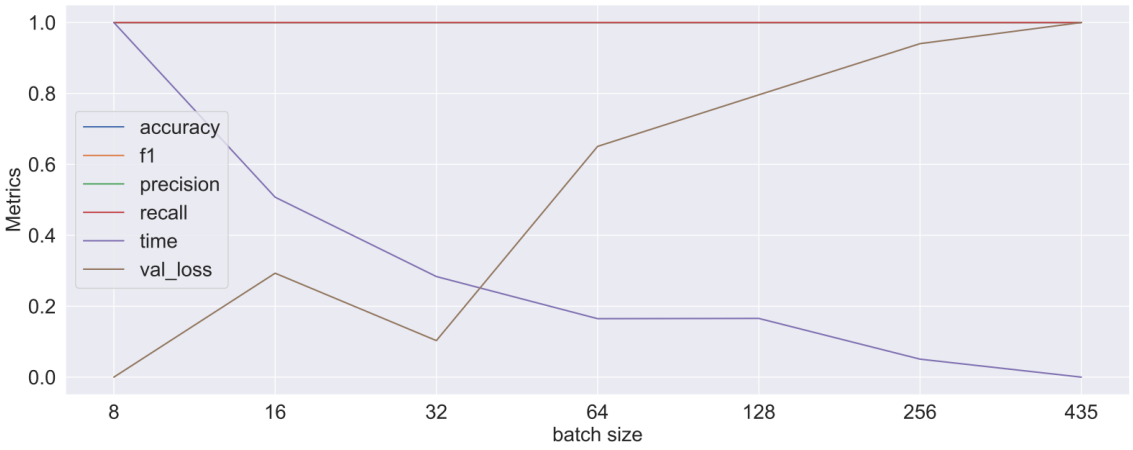


Figure 5.9: LSTM Auto-Encoder metrics with batch variation

5.3.1.3 Optimizing number of neurons

We also varied the architecture of the LSTM Auto-Encoder by changing the number of neurons in the hidden layers. A higher number of neurons contributes to a more

complex Auto-Encoder, being able to represent a higher number of characteristics of the dataset, but it can also overfit the model with the training data. We tried three different configurations, in configuration 0 we used 8 neurons in the first hidden layer and 4 in the second, in configuration 1 we used we used 16 neurons in the first hidden layer and 8 in the second and in configuration 2 we used 32 neurons in the first hidden layer and 16 in the second. For this case, we also fixed the value of timesteps at 7, the batch number at 32, and we used 'ReLU' activation function. In figure 5.10 we can see that configuration 0 presents the best results in terms of f1-score and it also took the least amount of time to train and test. All the configurations found anomalies during the time of the complaint, but configuration 2, since it has higher complexity, found more anomaly points outside the complaint time, meaning that probably the model was overfitting the training data.

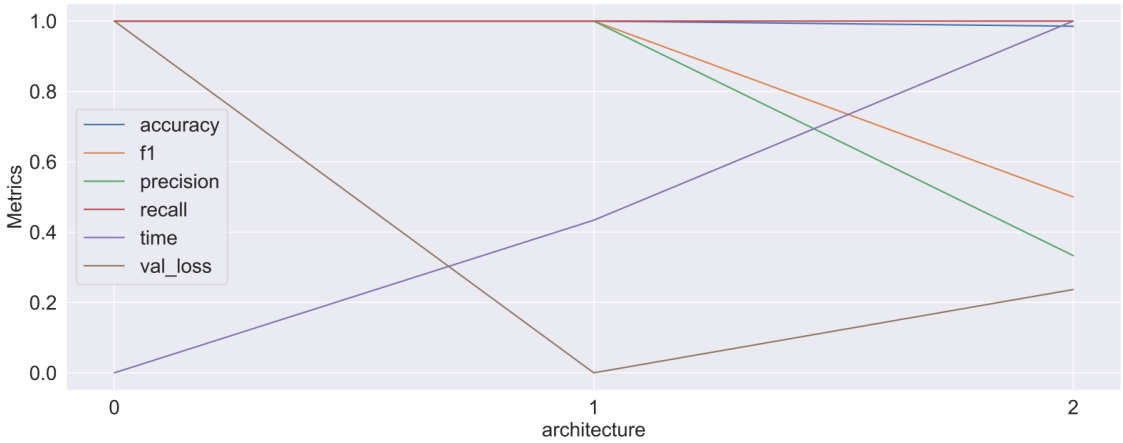


Figure 5.10: LSTM Auto-Encoder metrics with architecture variation

5.3.1.4 Implemented architecture

To observe specifically the impact of each parameter on its own, in the previous subsections, we varied the parameters one by one to see their influence on the reconstruction of the data and subsequent anomaly detection performance. Doing tuning in this way, we did not test all the possible parameter configurations. Nevertheless, based on the results of the previous sections, we decided to use 7 as the timestep value and the configuration 0, which uses 8 neurons in the first layer and 4 in the second. The constructed LSTM Auto-Encoder is demonstrated in figures 5.11 and 5.12. In the encoder segment, we have in the first layer an LSTM with 8 neurons, that receives as input data in the shape of 7 timesteps and 11 attributes and outputs in the shape of 7 timesteps and 8 attributes. This layer is configured to output a sequence, and therefore it outputs all timesteps. The second layer has 4 neurons and receives as input the output from the previous layer, but outputs in the shape of 1 timestep and 4 attributes, since it is configured to output only the last timestep as a result. The output of the encoder is then in the shape of 1 timestep and 4 attributes, and this represents the encoded attributes as demonstrated in figure 5.11. After the encoder segment, we have a third layer composed of a repeat vector, that transforms

the encoded attributes to the shape of 7 timesteps and 4 attributes by simply repeating the encoded attributes 7 times to get our data in the chosen number of timesteps. After the repeat vector, we have the decoder segment, which contains the fourth layer, which is an LSTM with 4 neurons. This layer receives as input the output of the repeat vector, as demonstrated in figure 5.12, and outputs in the same shape as the input. It is configured to output sequences, so every timestep outputs a result, and the layer has 4 neurons so it outputs also 4 attributes. Then the fifth layer is an LSTM network with 8 neurons and configured to return sequences. It receives as input the output from the previous layer and outputs 7 timesteps and 8 attributes. Lastly, we have a time distributed layer that transforms the previous output, in the output of 7 timesteps and 11 attributes has we had originally, by performing a matrix multiplication as explained in figure 5.12.

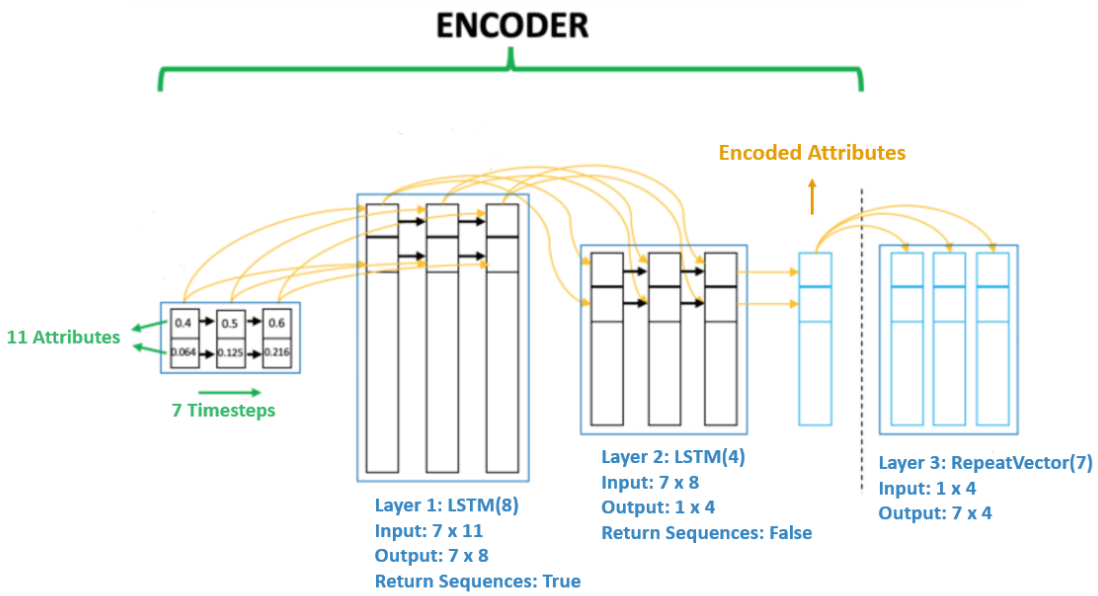


Figure 5.11: LSTM encoder architecture [36]

After we had the LSTM Auto-Encoder implemented, we used the data points from 03/09/2019 to 23/09/2019 of 2,000 customers with good scores and no complaints to train the LSTM Auto-Encoder. During the parameter tuning we were using data from only one customer from 03/09/2019 to 23/09/2019 to train the neural network and then tested it on the data from that one customer between 23/09/2019 and 30/09/2019. Now, we are using data from 03/09/2019 to 23/09/2019 belonging to multiple customers, building this way a general model with information of multiple customers to train and then we applied it individually to the other customers. For LSTM Auto-Encoder training, we used 2,000, and not 50, as we did in the software, because LSTM neural networks need high quantity of data to be trained to avoid overfitting the model. Also, some algorithms from the software take a very long time to run on large datasets, so it was not possible to use many customers for the training, since each customer had 11 measurements from

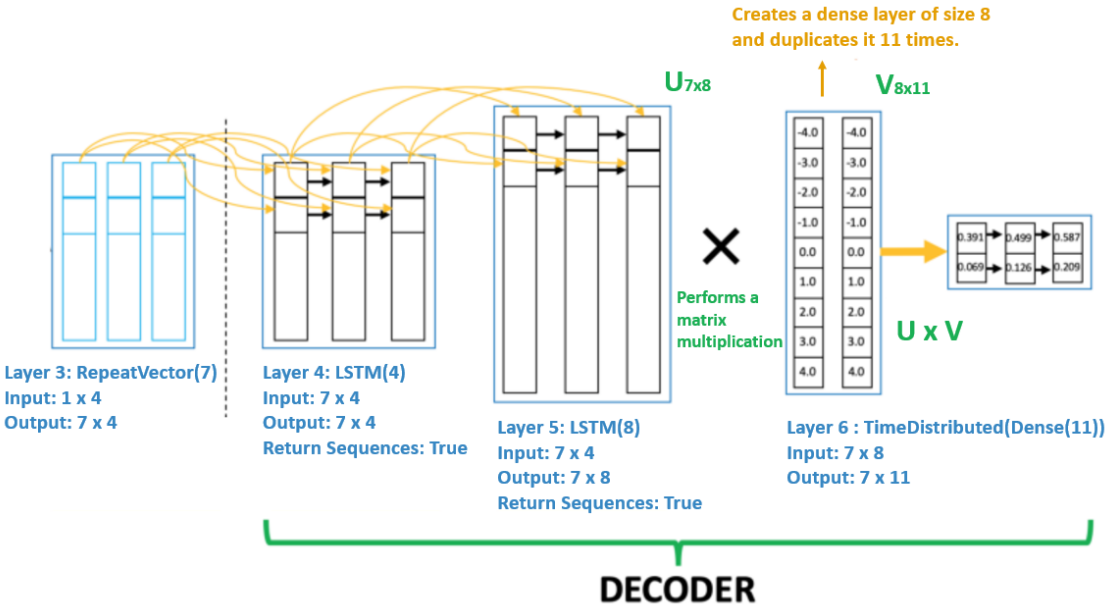


Figure 5.12: LSTM decoder architecture [36]

21 hours per day, for 3 weeks. All the customers' data was divided into 435 blocks of 7 timesteps, and the batch size was set to 435, so that in each iteration, network is trained with data from a different customer, and each batch only contains data points from a single customer. The activation function used in the LSTM layers was 'Relu'. After the Auto-Encoder was trained with the data from the 2,000 customers, it was tested on 30 customers with complaints, using the data from 24/09/2019 to 30/09/2019. To compare with the results from the software we used the same adjusted labeling, where if the LSTM Auto-Encoder classifies as anomaly at least one of the 11 data points with anomaly label it counts as one true positive, and if the LSTM Auto-Encoder does not classify as anomaly any of those 11 data points, it is counted as one false negative. All the normal data points classified as an anomaly are considered false positives.

5.3.2 Results

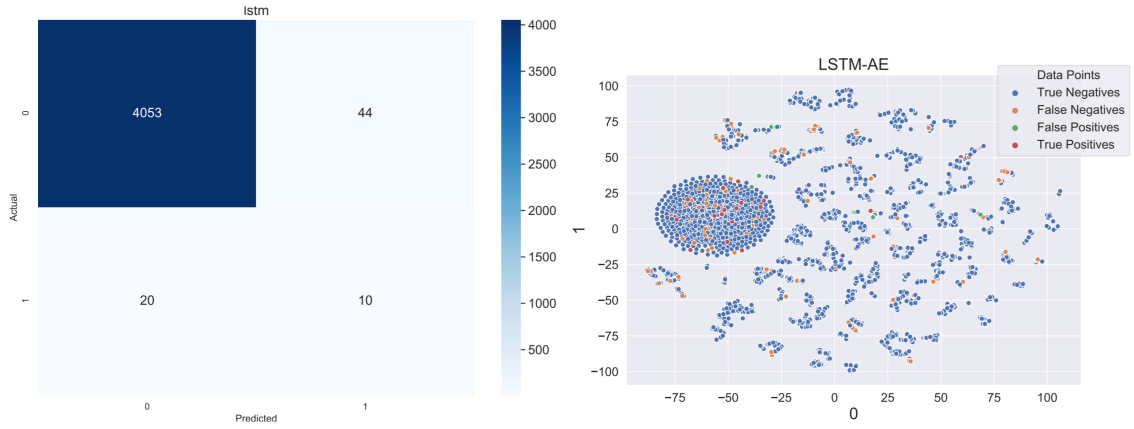
After the LSTM Auto-encoder was trained, it was applied to the same testing dataset as the anomaly detection software. In table 5.3, we have the results for different performance metrics for the LSTM Auto-Encoder and from the 3 best algorithms from the software in terms of f1-score: One-Class SVM, PCA, and Elliptic Envelope.

Time

In terms of execution time, we can see that the LSTM Auto-Encoder had results of the same magnitude as the majority of the software algorithms, although these results are not really comparable, since on one hand, LSTM Auto-Encoder used much more data than the other algorithms, and on the other, this time does not include the parameter tuning of LSTM, whereas tuning is included in the software execution time.

F1-Score, Recall and Precision

LSTM Auto-Encoder is superior in terms of f1-score in relation to the best algorithms of the software. It also has a higher precision, but One Class SVM and Elliptic Envelope achieved better recall. This means that both One-Class SVM and Elliptic Envelope found more customers with anomalies but at the cost of having more false positives. We can see in figure 5.13a that the LSTM Auto-Encoder found at least 1 anomaly in the time of complaint of each one of 10 customers and had 44 false positives. In figure 5.14 we show an example of a customer with a found anomaly by the LSTM Auto-Encoder on the period of the complaint. In this figure we have the reconstruction error of one customer in the period of 7 days, and the specific threshold for that anomaly score. As we can see, although the customer has anomalies on day 25 and 26, their values are not sufficiently large to be considered as an anomaly.



a LSTM Auto-Encoder confusion matrix

b LSTM Auto-Encoder results t-SNE

Figure 5.13: LSTM Auto-Encoder results for the quality of service dataset

When comparing these values with the values from figure 5.2a, we can conclude that One-Class SVM found 12 anomalies, but it also had more false positives. It is also possible to observe in figure 5.13b that the majority of the true positives found by LSTM Auto-Encoder are located in the bigger cluster containing the imputed missing values data points. The same testing data points were used in the software and for the LSTM Auto-Encoder, though t-SNE plots seem slightly different. This could be due to the ordering of the data points per customer in the case of LSTM, which led to small change in positions. Still, the clusters comprise data points of the same customers and the bigger cluster has the data points with missing values.

As we explained before, the results for lift, ROC-AUC, and PR-AUC are averages of the results of each customer, because the threshold was set individually for each customer. The mean lift in the first percentile of the LSTM Auto-Encoder is vastly superior to the mean lifts of the software algorithms, meaning that LSTM Auto-Encoder is better at attributing highest anomaly scores to real anomalies than the other algorithms, at least for the most of the tested customers. Despite that, in the first decile, its results are

equivalent to the ones of the other algorithms. Also in terms of PR-AUC, the results for the LSTM Auto-Encoder are superior to the other ones from the software, meaning that it achieves a better equilibrium between precision and recall as we have seen before.

Table 5.3: Comparison between LSTM-AE and three of the best algorithms from the anomaly detection software

Algorithm	Time	Mean Lift_1	Mean Lift_10	Mean Roc_Auc	Mean PR_Auc
lstm_AE	853	41.17	4.21	0.676	0.353
one_class_SVM	6032	9.13	4.57	0.742	0.153
pca	1	18.33	3.87	0.690	0.312
elliptic_envelope	45	31.96	4.22	0.688	0.283

Algorithm	F1	Recall	Precision	Accuracy
lstm_AE	0.238	0.333	0.185	0.985
one_class_SVM	0.216	0.400	0.148	0.979
pca	0.213	0.333	0.156	0.982
elliptic_envelope	0.203	0.400	0.136	0.977

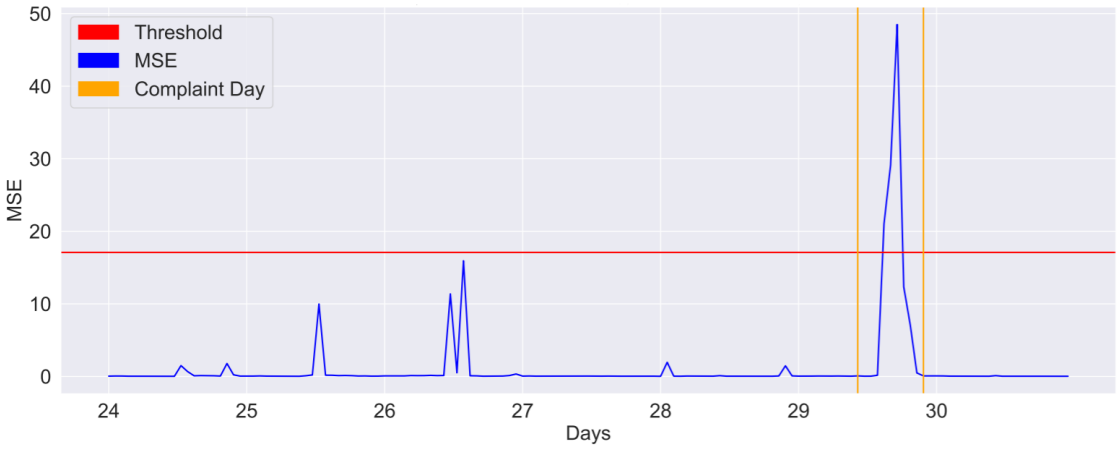


Figure 5.14: Example of customer with an anomaly found in the hours of complaint

5.3.3 Impact of Contextual Variables and Treatment of Missing Values

In this section, we analyze the impact of contextual variables on the detection of anomalies, and also how a different technique of missing values imputation could change the results obtained relative to the first technique. For this analysis, we kept the original architecture explained in the above section, and the only change was the addition of

contextual attributes and a different technique for missing value imputation. Same as before, we used the data from the first 3 weeks from the same 2,000 customers of the above section to train the LSTM Auto-Encoder. The model was then applied to the last week of the 30 customers with complaints in that week, as it was done previously.

The contextual variables we added were: hour and day of the week of the measurement, model of the router and CMTS the router connects to. The objective of this contextual information is to provide additional information which might help LSTM Auto-Encoder reconstruct the points. Since there is a great difference between the values of the attributes of different customers, learning the context could help in reconstructing the data points of that same context.

In the previous section, we imputed missing values by an extreme value, by *minmax* scaling of the dataset attributes between 0 and 1 and then replacing the missing values by -0.1. Now, we apply a new technique that consists of replacing the missing values by the mean of the specific attribute and then adding a flag attribute with the indication if that data point was imputed.

Next, we present the results for the 30 customers using the contextual information and imputation by mean, and then we analyze the impact that these modifications have in terms of customers' anomaly scores. The setup we used previously, in section 5.3.1.4, will be referred to in this section as the standard LSTM setup.

Contextual Variables

We tested 7 different configurations of contextual information:

- day of the week the data point was recorded;
- hour the data point was recorded;
- day of the week and the hour the data point was recorded;
- CMTS of the customer;
- model of the router device of the customer;
- CMTS and the model of the customer;
- all of the above

For all these configurations, we kept the same missing value imputation model as in the standard LSTM-AE setup. In table 5.4 we can see the changes in the f1-score, recall, precision, and accuracy resulting from addition of contextual information. Although none of the different configurations achieved a better f1-score than the original setup, three of the context variable configurations: using the day of the week, using the day of the week plus the hour, and using the CMTS, achieved better recall, meaning that they found more anomalies but at the cost of having more false positives, thereby achieving lower precision.

Table 5.4: Comparison of different LSTM-AE contextual configurations

Context	F1	Recall	Precision	Accuracy
None	0.238	0.333	0.185	0.985
Day	0.180	0.367	0.120	0.976
Hour	0.205	0.300	0.155	0.983
Day/Hour	0.197	0.433	0.127	0.974
CMTS	0.189	0.400	0.124	0.975
Model	0.204	0.333	0.147	0.981
CMTS/Model	0.180	0.267	0.136	0.982
All	0.185	0.333	0.128	0.979

Next, we analyze the customers where the contextual configurations successfully found the anomalies that the standard LSTM-AE setup could not. For that, we select 3 different customers from the testing dataset, where the contextual models were superior. For each customer we show a plot of the mean squared error (MSE) that represents the anomaly score of that customer during 7 days used for test, the threshold for that customer, and the 11-hour period that surrounds the complaint hour denoted as two red vertical lines. In each case, we will show the information from the standard LSTM-AE setup, as well as the setup with contextual information, and the information from these two models is shown with a different color.

The plot in figure 5.15 represents a customer with a complaint on day 24. As we can see from the figure, the MSE generated by the standard LSTM-AE model, represented in blue, did not go above the threshold line at the time of the complaint, hence that anomaly was not identified. MSE has several spikes, and they mostly correspond to data points with some missing values. Since LSTM Auto-Encoder uses sequences of data points to predict the anomaly scores, the biggest spike, on day 26 corresponds to a point which has many missing values, but it is preceded mostly by points where only some attributes were missing. This resulted in a larger reconstruction error than in the case of a reconstruction of data points around the complaint, which also had many missing values, but they were also preceded with points that similarly had many missing values. However, when contextual information was added, in this case, the day of the week, LSTM Auto-Encoder successfully identified an anomaly in that period, but at the cost of having 5 false positives for day 25.

The plot in figure 5.16 represents a customer with a complaint on day 26. The reason for this complaint is probably a malfunction in the device that was then replaced because after the day of the complaint there were no available measurements for this device. The anomaly score generated by the standard LSTM-AE setup, represented in blue, did not go above the threshold line ever during those 7 days and this is because the LSTM

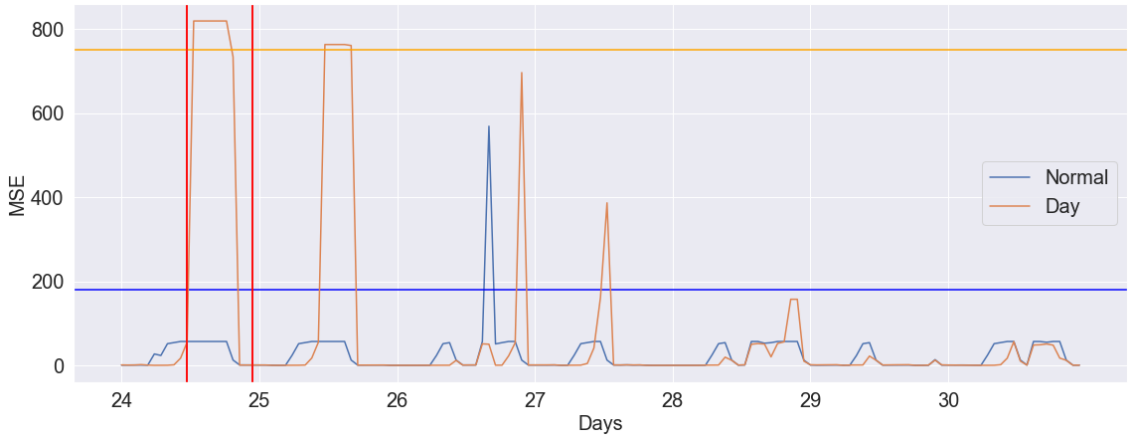


Figure 5.15: Resultant anomaly score and threshold when using the contextual information of the day of the week

Auto-Encoder could not reconstruct well the missing data after the day of the complaint and attributed high anomaly scores during that entire period elevating the threshold. When using the contextual information of the day of the week and hour of the day the LSTM Auto-Encoder successfully found an anomaly in that period, even though it had the similar value of the reconstruction error as the original model for those points. However, LSTM Auto-Encoder learned to reconstruct consequent missing data points, and so the threshold was much lower.

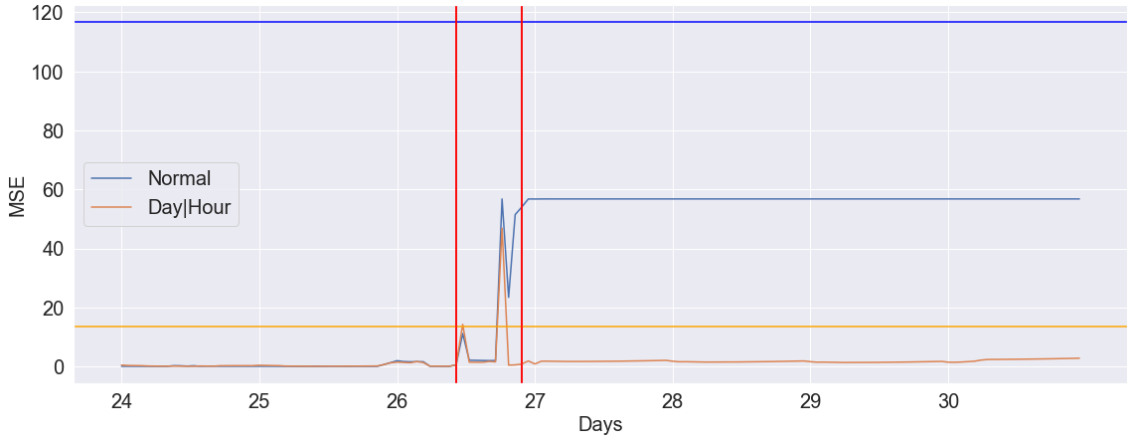


Figure 5.16: Resultant anomaly score and threshold by using the contextual information of the day of the week and hour

The last example, shown in figure 5.17, represents a customer with a complaint on day 25. This is also a customer with a lot of missing values in a sequence, and even though the standard LSTM-AE model gave high anomaly score to the points around the time of the complaint, the quantity of data points with a high anomaly score elevated the value of the threshold, thereby, not resulting in any anomaly identification. By using the contextual information of the customer's CMTS, LSTM Auto-Encoder learned to reconstruct the

large sequence of missing values, and it gave higher anomaly scores only to the points in the transition zones between missing values and present values.

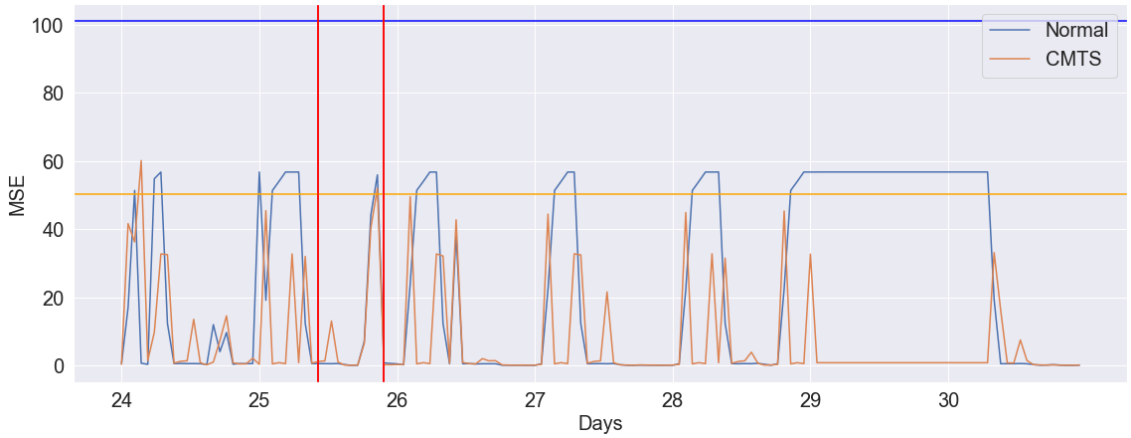


Figure 5.17: Resultant anomaly score and threshold by using the contextual information of the customer's CMTS

Missing Value Imputation

Next, we tested a different technique for missing values: we replaced them with the mean of the respective attribute, and then we added a flag attribute with the indication whether that data point was imputed. We compared the results obtained from LSTM Auto-Encoder trained with this new dataset and compared them with the results from the standard model. In table 5.5 we have a comparison of the two imputation methods. The extreme value method, used in standard LSTM-AE setup, results in a higher f1-score and precision, but the mean plus imputation flag method gave a higher recall, meaning that it identified more anomalies.

Table 5.5: Comparison of LSTM-AE results obtained by using different missing value imputation methods

Missing Value Imputation	F1	Recall	Precision	Accuracy
Extreme Value	0.238	0.333	0.185	0.985
Mean + Imputation Flag	0.202	0.367	0.139	0.979

We show an illustrative example of a case when the mean plus imputation flag method led to superior performance. In figure 5.18 we show the anomaly score of the same customer shown in figure 5.17, but this time we compare the results of the standard LSTM-AE setup with the results obtained by using imputation by mean and addition of a new flag indicator variable. In the standard model, where missing values were substituted with an extreme value, the threshold for the anomaly score was set too high, and the reconstruction error was not large enough for any data point to intercept it, as explained before. With the other method, all the missing values were replaced by the

mean of that attribute, hence the LSTM Auto-Encoder did not attribute them a very high anomaly score, detecting only as anomalies the data points near the time of complaint. Around the time of the complaint, the value of the attribute upstream corrected codeword error increased from 0 to 33, which led to high error, corresponding to the orange spike on the plot, and subsequently, to correct identification of the anomaly.

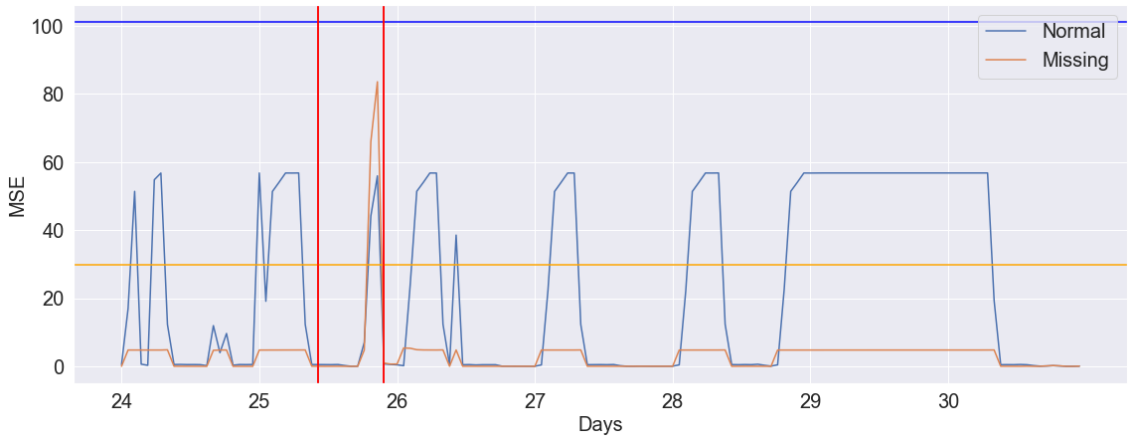


Figure 5.18: Resultant anomaly score and threshold by using imputation of missing values by mean and addition of flag

5.4 Conclusions

In this chapter, we approached the difficult problem of trying to detect periods when customers of a telecommunication company experienced low quality of internet service. As explained before, this is a challenging problem since there are many different reasons for which a customer could make a complaint regarding service quality, and the variables that reflect those reasons might not be present in our dataset. Furthermore, the labels used to train and evaluate the models are not completely exact, since there can be a time lag between the problem happening and the customer making the complaint. By comparing the results from the developed software with the results of a state of art time series anomaly detection algorithms we conclude that it is beneficial to use the sequential information present in the data. In this way, anomalies are detected not only based on differences in values of attributes of different data points, but also considering the attributes of points adjacent in time. One important conclusion of this chapter is that different values of parameters of the LSTM Auto-Encoder impact the way that the algorithm learns to reconstruct new data points. In the case of our dataset we concluded that a lower timestamp, higher batch size and simpler architecture gave better results not only in the performance metrics but also on the time consumption of the algorithm. We also analyzed the impact of adding contextual variables and using different treatment of missing values on the performance of LSTM-AE. We observed that although there was

not a major improvement in the performance metrics, these modifications changed the way that the algorithm learned to represent the data.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The main objective of this dissertation was to address different problems of a telecommunication company with anomaly detection approaches. To achieve this objective, we developed a general software that could be applied to any type of anomaly detection problem, and that could be used to understand which algorithms and what configuration of parameters would be the best for each specific problem. The software first performs necessary data pre-processing, such as data scaling and treatment of missing values. Then it optimizes ten different anomaly detection models and applies them to the given dataset. As output, it provides the results of all the algorithms, their different performance metrics and the values of the optimized parameters. Additionally, it generates t-SNE plots showing anomalies found by each algorithm. These two-dimensional plots of high-dimensional data provide the user with a visual aid to understand the characteristics of the data points being classified as anomalies. Furthermore, the software not only calculates the traditional performance metrics of the algorithms' performance, for example, precision, recall, f1, but it also evaluates two special metrics: Excess-Mass and Mass-Volume. These two metrics allow performance comparison of algorithms even in the case when no labels are available, i.e., when the ground truth on which points are anomalies is not known. After applying the software on the dataset of interest, the user can select the best algorithm and the best parameters, based on the metrics most suitable for a given business problem.

We applied this software to 3 different business cases with different characteristics between them: detection of fraudulent transactions, identification of clients likely to miss their payment and detection of periods of low quality of Internet service. In each of these three business cases we used a real world dataset, and these datasets had an increasing levels of difficulty associated with them. We applied the software, and discussed its

results in detail, explaining anomaly detection performance of the algorithms by taking into account the business problem in question. In the case of fraudulent credit card transactions, where the dataset was the least complex, almost all the algorithms of the software achieved similar results. In the case of identification of clients likely to miss their payment, which was more challenging that the previous, distance-based algorithms, such as Local Outlier Factor and K-Nearest Neighbor, and the algorithms that function by capturing the distribution of the normal data, such as One-Class SVM, Gaussian Mixture Models and Elliptic Envelope performed much better than the ones based on isolation and dimensionality reduction. For the third business case, where we wanted to identify when did Internet clients experienced low quality of service, we had the most challenging dataset. There were many missing values in it, we did not have the exact labels, and the data was in fact a multivariate time-series. In this case, the best algorithms were the ones that capture the distribution of normal data, such as One-Class SVM and Elliptic Envelope, as well as the ones based on dimensionality reduction: Auto-Encoder and PCA. However, none of these algorithms had a very good performance, due to the problem complexity, as well as the fact that they were not considering the time-series aspect of the data. Overall, looking at the results obtained in these three cases, we conclude that there is not a single algorithm, or a parameter configuration that is suitable for all business cases and all the performance metrics used.

We also implemented a state of the art architecture of a time series anomaly detection algorithm, LSTM Auto-Encoder, that was used again to address the problem of identification of low QoS, but this time, by taking advantage of the sequential nature of measurements. We analyzed the impact of LSTM Auto-Encoder parameters on its performance for our specific dataset. We observed that a lower value of timestamp, higher batch size and simpler architecture gave better results, not only regarding the performance metrics, but also the algorithm's running time. When we compared the performance of the LSTM Auto-Encoder that was using optimized parameters with the performance of the algorithms from the software, we observed that it achieved better results than all the algorithms in most of the performance metrics. Furthermore, we also analyzed whether an addition of different contextual variables and a change in the treatment of missing values would improve the results of the LSTM Auto-Encoder. We observed that some contextual attributes could help LSTM Auto-Encoder find more anomalous points, though this was not true for all the tested cases. Regarding the change in the treatment of missing values, we concluded that replacing the values with the mean, and then adding a flag informing whether the data point was imputed also led in some cases to LSTM Auto-Encoder being less sensitive to missing values, allowing it to give more importance to the variations in values that are not missing, thereby identifying more anomalies.

6.2 Future Work

Some improvements could be made to the two main contributions of this dissertation, the anomaly detection software, and the LSTM Auto-Encoder for time-series data.

Anomaly Detection Software:

- Incorporating more anomaly detection methods to the software, such as DBSCAN and robust PCA.
- Adding the option of combining different detection approaches into ensembles, which might improve the results as different anomaly detection algorithms find anomalies with different characteristics.
- Parametrizing the software even more, by adding an option of allowing the user to select a subset of algorithms they want to use, instead of running all of the algorithms.

LSTM Auto-Encoder:

- Initializing the LSTM Auto-Encoder state using contextual variables, by adding another layer to the architecture.
- Adding a dropout layer to make the network more robust to missing values in the data.
- Pre-training the network with synthetic data with known, 100% certain, artificial labels.
- Imputing the missing values by mean, adding the flag with the value 1 if the data point was imputed, and then randomly changing some 0 values of the flag attribute to 1 in order to generalize the model and lower the importance of the missing values.

Bibliography

[1] T. Bouwmans and E. H. Zahzah. "Robust PCA via principal component pursuit: A review for a comparative evaluation in video surveillance." In: *Computer Vision and Image Understanding* 122 (2014), pp. 22–34.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. "LOF: identifying density-based local outliers." In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104.

[3] R. Chalapathy and S. Chawla. "Deep learning for anomaly detection: A survey." In: *arXiv preprint arXiv:1901.03407* (2019).

[4] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly detection: A survey." In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[5] *Credit Card Fraud Detection*. https://www.kaggle.com/mlg-ulb/creditcardfraud.

[6] J. Davis and M. Goadrich. "The relationship between Precision-Recall and ROC curves." In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 233–240.

[7] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui. "A comparative evaluation of outlier detection algorithms: Experiments and analyses." In: *Pattern Recognition* 74 (2018), pp. 406–421.

[8] *Extended Isolation Forest (EIF)*. https://github.com/sahandha/eif.

[9] H. Goel, I. Melnyk, N. Oza, B. Matthews, and A. Banerjee. "Multivariate Aviation Time Series Modeling: VARs vs. LSTMs." In: ().

[10] N. Goix. "How to evaluate the quality of unsupervised anomaly detection algorithms?" In: *arXiv preprint arXiv:1607.01152* (2016).

[11] M. Goldstein and S. Uchida. "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data." In: *PloS one* 11.4 (2016), e0152173.

[12] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[13] J. Hardin and D. M. Rocke. "Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator." In: *Computational Statistics & Data Analysis* 44.4 (2004), pp. 625–638.

[14] S. Hariri, M. C. Kind, and R. J. Brunner. "Extended isolation forest." In: *arXiv preprint arXiv:1811.02141* (2018).

[15] S. Hawkins, H. He, G. Williams, and R. Baxter. "Outlier detection using replicator neural networks." In: *International Conference on Data Warehousing and Knowledge Discovery*. Springer. 2002, pp. 170–180.

[16] V. Hodge and J. Austin. "A survey of outlier detection methodologies." In: *Artificial intelligence review* 22.2 (2004), pp. 85–126.

[17] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom. "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding." In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 387–395.

[18] X. Jin, Y. Zhang, L. Li, and J. Hu. "Robust PCA-based abnormal traffic flow pattern isolation and loop detector fault detection." In: *Tsinghua Science & Technology* 13.6 (2008), pp. 829–835.

[19] T. Kieu, B. Yang, and C. S. Jensen. "Outlier detection for multidimensional time series using deep neural networks." In: *2018 19th IEEE International Conference on Mobile Data Management (MDM)*. IEEE. 2018, pp. 125–134.

[20] H.-P. Kriegel, M. Schubert, and A. Zimek. "Angle-based outlier detection in high-dimensional data." In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 444–452.

[21] F. T. Liu, K. M. Ting, and Z.-H. Zhou. "Isolation forest." In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE. 2008, pp. 413–422.

[22] *LSTM Input Shape*. https://shiva-verma.medium.com/understanding-input-and-output-shape-in-lstm-keras-c501ee95c65e. Accessed: 25-11-2020.

[23] L. v. d. Maaten and G. Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[24] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. "LSTM-based encoder-decoder for multi-sensor anomaly detection." In: *arXiv preprint arXiv:1607.00148* (2016).

[25] P.-F. Marteau, S. Soheily-Khah, and N. Béchet. "Hybrid Isolation Forest-Application to Intrusion Detection." In: *arXiv preprint arXiv:1705.03800* (2017).

[26] T. P. Minka. "Automatic choice of dimensionality for PCA." In: *Advances in neural information processing systems*. 2001, pp. 598–604.

[27] J. Muñoz-Marí, F. Bovolo, L. Gómez-Chova, L. Bruzzone, and G. Camp-Valls. "Semisupervised one-class support vector machines for classification of remote sensing data." In: *IEEE transactions on geoscience and remote sensing* 48.8 (2010), pp. 3188–3197.

[28] S. Omar, A. Ngadi, and H. H. Jebur. "Machine learning techniques for anomaly detection: an overview." In: *International Journal of Computer Applications* 79.2 (2013).

[29] R. Paffenroth, K. Kay, and L. Servi. "Robust pca for anomaly detection in cyber networks." In: *arXiv preprint arXiv:1801.01571* (2018).

[30] S. Ramaswamy, R. Rastogi, and K. Shim. "Efficient algorithms for mining outliers from large data sets." In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 427–438.

[31] A. Reddy, M. Ordway-West, M. Lee, M. Dugan, J. Whitney, R. Kahana, B. Ford, J. Muedsam, A. Henslee, and M. Rao. "Using gaussian mixture models to detect outliers in seasonal univariate network traffic." In: *2017 IEEE Security and Privacy Workshops (SPW)*. IEEE. 2017, pp. 229–234.

[32] P. J. Rousseeuw and K. V. Driessen. "A fast algorithm for the minimum covariance determinant estimator." In: *Technometrics* 41.3 (1999), pp. 212–223.

[33] H. Sak, A. Senior, and F. Beaufays. "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition." In: *arXiv preprint arXiv:1402.1128* (2014).

[34] M. Sakurada and T. Yairi. "Anomaly detection using autoencoders with nonlinear dimensionality reduction." In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. 2014, pp. 4–11.

[35] *Understanding input and output shape LSTM*. https://towardsdatascience.com/lstm-autoencoder-for-extreme-rare-event-classification-in-keras-ce209a224cfb. Accessed: 25-11-2020.

[36] *Understanding LSTM Auto-Encoder Layers*. https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ffab055b6352. Accessed: 25-11-2020.

[37] R. Wirth and J. Hipp. "CRISP-DM: Towards a standard process model for data mining." In: *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*. Citeseer. 2000, pp. 29–39.

[38] Y. Zhao, Z. Nasrullah, and Z. Li. "PyOD: A Python Toolbox for Scalable Outlier Detection." In: *Journal of Machine Learning Research* 20.96 (2019), pp. 1–7. URL: http://jmlr.org/papers/v20/19-011.html.

APPENDIX A

t-SNE PLOTS WITH PREDICTIONS OF THE ANOMALY DETECTION SOFTWARE ALGORITHMS

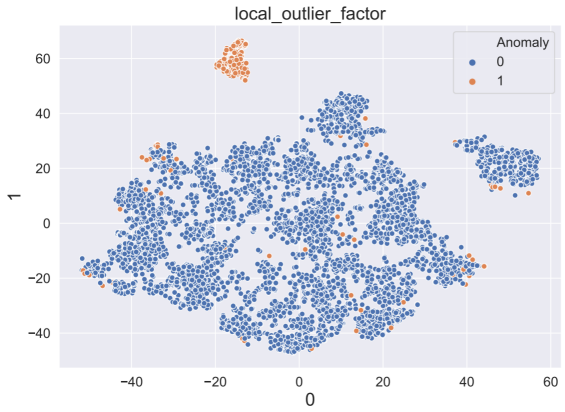
In this appendix are the t-SNE plots with the results of the found anomalies of the different algorithms that were not shown for the two datasets analyzed in chapter 4. They can be compared to the t-SNE plots of the real anomalies for each one of the datasets found on figure 4.2 and 4.11a.

Credit Card Dataset

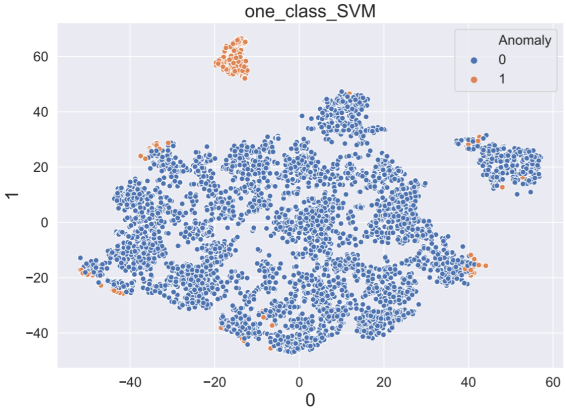
(a) t-SNE with predictions of Isolation Forest

(b) t-SNE with predictions of Extended Isolation Forest

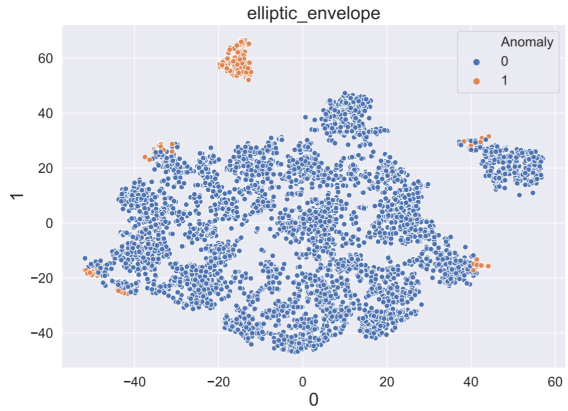
Figure A.1: t-SNE plots with the predictions of the tuned algorithms for the credit card dataset



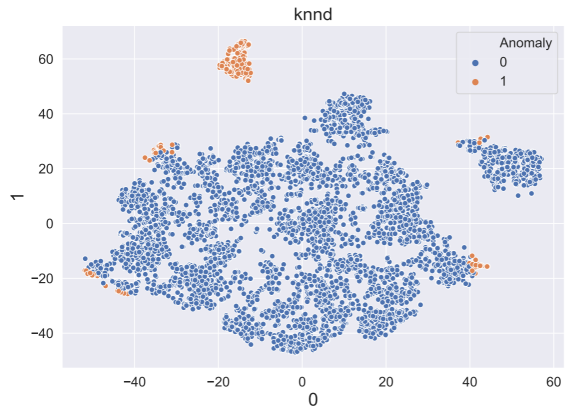
(c) t-SNE with predictions of Local Outlier Factor



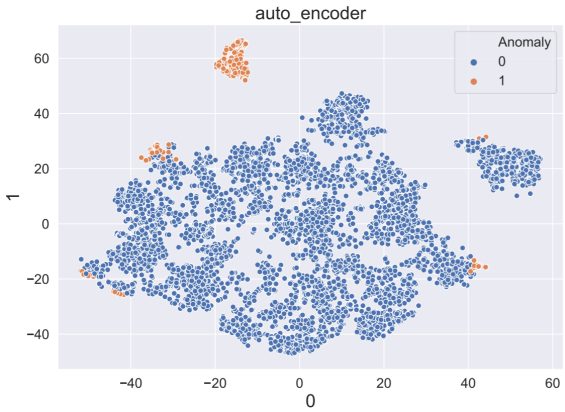
(d) t-SNE with predictions of One-Class SVM



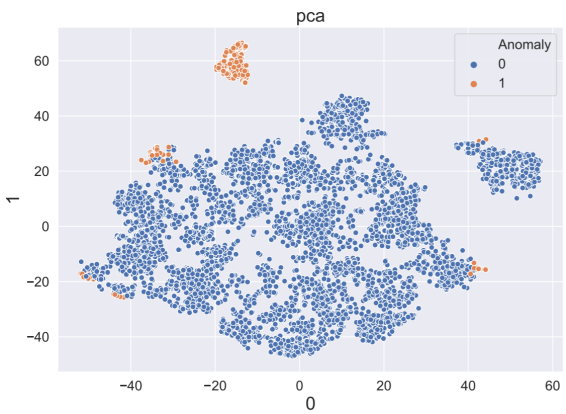
(e) t-SNE with predictions of Elliptic Envelope



(f) t-SNE with predictions of K-Nearest Neighbor Detector



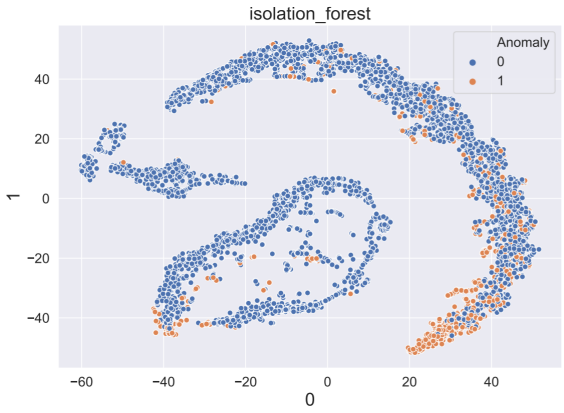
(g) t-SNE with predictions of Auto-Encoder



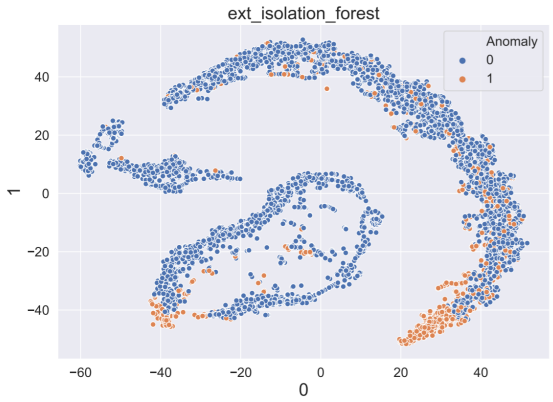
(h) t-SNE with predictions of Principal Component Analysis

Figure A.1: t-SNE plots with the predictions of the tuned algorithms for the credit card dataset

Involuntary Churn Dataset

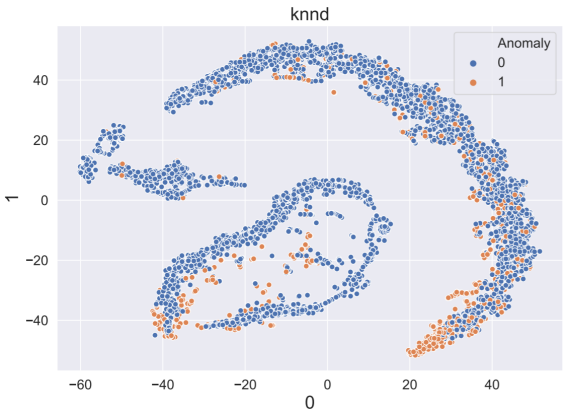


(a) t-SNE with predictions of Isolation Forest

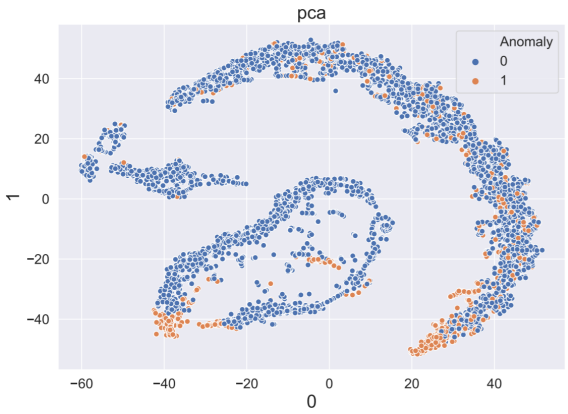


(b) t-SNE with predictions of Extended Isolation Forest

(c) t-SNE with predictions of One-Class SVM



(d) t-SNE with predictions of K-Nearest Neighbor Detector



(e) t-SNE with predictions of Principal Component Analysis

Figure A.2: t-SNE plots with the predictions of the tuned algorithms for the involuntary churn dataset